Knowledge-Based Mining of Multi-Databases for Associations

A Thesis

Submitted for the Degree of

Doctor of Philosophy

in the Faculty of Engineering

by

Ananthanarayana V. S.

Department of Computer Science and Automation

INDIAN INSTITUTE OF SCIENCE

BANGALORE-560 012, INDIA

JUNE 2001

Knowledge-Based Mining of Multi-Databases for Associations

A Thesis

Submitted for the Degree of

Doctor of Philosophy

in the Faculty of Engineering

by

Ananthanarayana V. S.

Department of Computer Science and Automation

INDIAN INSTITUTE OF SCIENCE

BANGALORE-560 012, INDIA

JUNE 2001

Abstract

Knowledge discovery in databases (KDD) uses knowledge engineering tools and database technology to extract hidden patterns from databases. Data mining is a step in the KDD process which finds useful patterns in the data. There are many categories of data mining. One of the most important of them is *association rule mining*. The association rule mining (ARM) activity establishes an association between two non-overlapping sets of frequently occurring values in a database. Most of ARM activity addressed in literature is based on *a single transaction database*.

In real world applications, we have a large collection of data that is organized in several databases. Mining relations coming out of different databases might reveal useful hidden associations. For example, the data about a person's behaviour is distributed across several organizations. More specifically, CUSTOMER database describes person's purchase behaviour; EMPLOYEE database gives his professional information; PATIENT database gives his medical history; and AIRLINE database gives PERSON's travel details. There may be an association between salary, region of living, mode of traveling and disease; like people who have salary in the range of US$ 10,000 - US$ 20,000, eat frequently at CITY X hotels and travel by air in executive class have cardio vascular disease, where CITY X is marked with a high pollution rating. These hidden associations can be discovered if we mine *across* multiple databases.

Most of the ARM algorithms are data-driven and hence generate a large number of associations. However, in practice, only a relevant subset of this rule-set, which aids in decision making, is of interest. Such a mining activity can be performed using *knowledge* for mining only those rules that are useful for decision making. Use of knowledge not only

reduces the number of associations generated but also reduces the computational time and space required for mining.

The research work presented in this thesis covers a spectrum of novel methods and data structures for efficient generation of associations. The following issues are addressed in this thesis :

- Multiple database-oriented ARM activity:

  Not enough attention is paid to mining multiple databases for associations. We call such associations, *inter-database associations.* In this thesis, we propose and implement efficient schemes to obtain inter-database associations. The proposed schemes use domain knowledge in the form of a semantic network to efficiently link various databases.

- Knowledge-based mining :

  In ARM activity, knowledge in the form of *is-a* hierarchy is used for discovering generalized rules. In this thesis, we propose a taxonomy based on functionality, and a restricted way of generalization of values and use it for efficient ARM.

- Single scan of the database :

  Data to be handled by the ARM activity is very huge, methods to handle this activity should be efficient in terms of usage of computational resources like disk and memory. The best possible algorithm reported in the literature requires *at least* two database scans. In this thesis, we propose schemes which need *a single* scan of the database for finding associations.

- Abstraction generation :

  In this thesis, we propose schemes that are based on intermediate representations (structures) of the data, which we call abstractions. These structures are generated based on data and/or domain knowledge. We use these abstractions instead of the data in the database for mining associations. Discovery of associations using

abstractions is more efficient because these structures represent data in *a compact* form and are *appropriate* for association finding.

- Dynamic mining :

  ARM activity based on change of data, called incremental mining and change of support value are described in the literature. Mining under *incremental change to knowledge* without having to access the original database is not reported in the literature. In this thesis, we propose novel schemes for mining which can handle change of data, knowledge and support value which we call *dynamic mining.*

In this thesis, we report four schemes. They are (i) Extended inverted file (EIF) based scheme, (ii) And-Or taxonomy (A O taxonomy) based scheme, (iii) Pattern Count Tree (PC-tree) based scheme and (iv) Multi-database Domain Link Network (MDLN) based scheme. Even though all the four schemes support discovery of associations across multiple databases, the PC-tree based and the A O taxonomy based schemes are basically single database oriented schemes.

Based on theoretical and experimental studies, we make the following observations :

- The MDLN structure based scheme is ideally suited for mining multiple databases for discovering associations.
- The scheme that exploits knowledge in the best possible way is the A O taxonomy based scheme. The knowledge helps in the reduction of number of candidates generated and number of database scans required.
- PC-tree based scheme is ideally suited for dynamic mining.

Acknowledgements

I wish to express my deep sense of gratitude and indebtedness to my research supervisors Prof. D.K. Subramanian and Prof. M. Narasimha Murty for their inspiring guidance, constructive comments and instructive suggestions. Their constant encouragement has inspired me to complete my work with ever sustaining enthusiasm and interest.

I wish to thank Prof. Y.N. Srikant, Chairman, Department of Computer Science and Automation for providing me various facilities of the department. I am thankful to all the faculty members of the department for their support and encouragement.

I am grateful to my parents for always encouraging me to do my best, and also to my wife, Anitha and son, Aniketh an who provided me the lovely-home atmosphere during my stay in the campus.

I wish to thank Mr. Gangaraju, Mrs. Lalitha, Mrs. Meenakshi and other staff of the department for their cheerful help at all times. Final word of thanks to all my friends whose presence has stimulated me to do my work with unbounded vigour and gusto.

List of Author's Publications Based on The Thesis

1. Ananthanarayana, V.S., Subramanian, D.K., Narasimha Murty, M. : *Scalable, Distributed and Dynamic Mining of Association Rules*, International Conference on High Performance Computing, December, 2000. (pp 559 - 566)

2. Subramanian, D.K., Ananthanarayana, V.S., Narasimha Murty, M. : *And-Or Concept Taxonomies for Mining Generalized Association Rules*, International Conference on Knowledge Based Computer Systems, December 2000. (pp 70 - 81)

3. Ananthanarayana, V.S., Narasimha Murty, M., Subramanian, D.K. : *Multi-Dimension Semantic Clustering of Large Databases for Association Rule Mining*, Pattern Recognition Journal, 2001, Vol 34, Issue 4. (pp 939 - 941)

4. Ananthanarayana, V.S., Narasimha Murty, M., Subramanian, D.K. : *An Incremental Data Mining Algorithm for Compact Realization of Prototypes*, Accepted for publication in Pattern Recognition Journal.

5. Subramanian, D.K., Ananthanarayana, V.S., Narasimha Murty, M. : *Mining Multiple Databases for Inter-Database Association Rules*, International Conference on Knowledge Based Computer Systems, December 2000. (pp 58 - 69)

6. Ananthanarayana, V.S., Narasimha Murty, M., Subramanian, D.K. : *Efficient Clustering of Large Data Sets*, Accepted for publication in Pattern Recognition Journal.

7. Ananthanarayana, V.S., Narasimha Murty, M., Subramanian, D.K. : *Mining for Frequent Valuesets (Itemsets): A Review from the Database Perspective*, communicated to Journal of Data Mining and Knowledge Discovery.

8. Ananthanarayana, V.S., Subramanian, D.K., Narasimha Murty, M. : *Distributed Abstraction Based Mining for Large Itemsets using a Single Database Scan*, communicated to The Information Systems Journal.

9. Ananthanarayana, V.S., Narasimha Murty, M., Subramanian, D.K. : *Mining Large Itemsets using a Single Database Scan*, communicated to Journal of Intelligent Information Systems.

10. Subramanian, D.K., Ananthanarayana, V.S., Narasimha Murty, M. : *An Efficient Mining of Dominant Entity Based Association Rules in Multi-databases*, communicated to The VLDB Journal.

11. Subramanian, D.K., Ananthanarayana, V.S., Narasimha Murty, M. : *Knowledge-Based Association Rule Mining using And-Or Taxonomies*, communicated to Knowledge Based Systems Journal.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 General Introduction

The past two decades have seen a huge increase in both the amount of data stored in databases and the number of business and scientific database applications. In early ages of database development, we had centralized databases for an organization. Due to the advances in distributed computing and networking, organizations moved towards *decentralizing* the processing (at the system level) while achieving an *integration* of information resources (at the logical level) within their geographically distributed systems of databases. With the schema integration methodology [Ananthanarayana, 95; Larson et.al, 86], it is possible to develop a system of federated databases which can be viewed as a collection of cooperative autonomous databases. Federated databases without any global schema constitute *multi-databases*. From the point of view of *local autonomy*, we have *distributed database* at one end and *multi-databases* at other end. Because of the technological developments, there is an increase in our capability of both collection and storage of data. As a consequence, scientific, government and corporate information systems are moving towards handling large databases amounting to hundreds of Tera or Pica bytes of data [Agrawal et.al, 92]. Knowledge engineering, on the other hand, focuses on development of techniques for inferring knowledge from data. *Knowledge Discovery in Databases (KDD)* uses knowledge engineering tools and database technology to extract

hidden patterns from a collection of databases. *Data mining* is a step in the KDD process which enumerates useful patterns (or models) over the data [Fayyad et.al, 96A].

Since data to be handled by the data mining activity is very huge, methods to handle this activity should be efficient in terms of usage of computational resources like memory and processing time. There are many categories of data mining activity [Fayyad et.al, 96B]. One of the important activities is association rule mining. In this thesis, we propose and implement schemes for efficient mining of association rules. In addition to the efficient discovery of knowledge from an organizational database, it is important to mine the knowledge which is hidden *across* organizational databases. For example, Health service department maintains a database corresponding to medical history of persons; while person's purchase information is available in transaction database maintained by supermarkets. An interesting question is "is it possible to find knowledge that relates a person's *purchase habits* with *his health status* ?" Answer to this question might emerge by - *mining across multiple databases.* This thesis deals with an efficient way of mining multiple databases for associations.

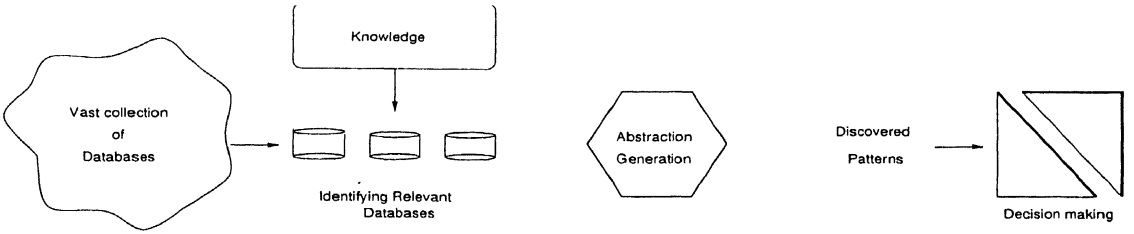
Remaining part of this chapter is organized as follows. In section 1.2, we give the description and steps involved in the KDD process. A general problem definition of data mining activity is given in section 1.3. Current status of association rule mining activity is explained in section 1.4. We give the motivation for our work in section 1.5 and the scope of the work reported in this thesis is given in section 1.6. A road map with a brief description of each chapter is given in section 1.7.

1.2 Knowledge Discovery in Databases (KDD)

Knowledge discovery in databases is an iterative and interactive process of discovering non-trivial, valid, previously unknown but potentially useful patterns or models [Fayyad et. al, 96B] from a collection of databases. Figure 1.1 illustrates the KDD process. Steps involved in the KDD process in the most general form are the following :

1. Identify a set of *relevant* databases based on the required attributes, from a vast

Figure 1.1 KDD process



collection of databases with the help of domain knowledge. This step is called *data selection.* This step may need the integration of heterogeneous multiple databases. Schema integration [Ananthanarayana, 95; Larson et.al, 86] can be used to handle this part of the step.

2. Reduce the search space by generating *abstractions* of the data present in the selected databases. This step involves *cleaning , transformation* and *reduction of data* in that order.

   - *Data cleaning :* This step filters out noisy, erroneous and irrelevant data.
   - *Data transformation and reduction :* This step involves finding useful features to represent data depending on the goal of the task. This may use dimensionality reduction or transformation methods to reduce the effective number of variables under consideration or to find invariant representations of the data and projecting the data onto spaces in which solution is likely to be easier to find.

     Data generalization is a kind of data transformation and reduction operation which abstracts a large set of relevant data in a database from a low concept level to relatively higher ones. For example, one may like to summarize a large set of the items related to some sales to give a general description.

3. Extract *relevant patterns* from the abstractions. This step is called *data mining.*

4. Use the knowledge discovered in manual or mechanical decision making.

A *Data warehouse* is a collection of databases that contains subject-oriented, integrated and historical data [Elmasri et.al, 00; Inmon, 96]. *Data warehousing* is the process of creating a data warehouse by collecting and cleaning historical data and making it available for mining and decision making. In the KDD process described above, the data warehousing could be identified with the transformation step (step 2) which just precedes the data mining step (step 3) of the process.

Data mining finds a variety applications. For example, many organizations are using data mining to help to manage customer relationships (i.e., CR Mining). By determining characteristics of customers who are likely to leave for a competitor, a company can act to retain that customer, because it is usually far less expensive to retain a customer than to acquire a new one.

In medicine, data mining can be used to find the relationship between eating habits and diseases, work environment and diseases, etc.

E-marketing is another area in which data mining can be useful. By identifying good candidates for mail offers or catalogs, direct mail marketer can reduce expenses and increase their sales. Targeting specific promotions to existing and potential customers can have a similar good effect. Other uses for data mining include : fraud detection and policy assessments in health and insurance; optimization, scheduling, web mining and process control.

There are two main forms of data mining [Simoudis, 96] :

1. *Verification-driven* data mining : where the user postulates a hypothesis, and the system tries to validate it. The common verification-driven operations include query and reporting, multidimensional analysis and statistical analysis.

2. *Discovery-driven* data mining : where the new information is extracted automatically. Different categories of discovery-driven task include :

- **Association rule generation :** Here, the aim is to find associations between patterns based on their frequency of co-occurrences in a tuple. The problem of mining association rules based on a transaction database is introduced in [Agrawal

et.al, 93A]. This kind of mining activity is aimed at market-basket analysis and it typically establishes an association between two non-overlapping sets of frequently bought items. An example of an association rule is : noodles $\Longrightarrow$ vegetable-oil, (σ = 4%, c = 88%). It means, 88% of transactions that contain *noodles* also contain *vegetable-oil*; 4% of all transactions contain both the items. Here, 88% is called the confidence of the rule (c), and 4% is the support (σ) of the rule. The problem is to find all of the association rules that satisfy the user-specified minimum support and confidence constraints.

Sequential Patterns : The problem of mining data for sequential patterns is closely related to the problem of mining for association rules. Sequence discovery aims at extracting sets of events that commonly occur over a period of time [Agrawal et.al, 95A]. An example of a sequential pattern could be that "80% of the people who buy volume 1 and 2 of Knuth's *The art of computing* also buy volume 3 within a year". Such patterns are useful in domains such as retail sales, telecommunications (eg. alarm diagnosis) and medicine (eg. identifying symptoms that precede diseases).

Similar Time Sequences : Time-series data constitute a large portion of data stored in computers. The capability to find time-series (or portions thereof) that are "similar" to a given time-series or to groups of similar time-series has several applications [Agrawal et.al, 93A; Faloutsos et.al, 94; Agrawal et.al, 95B]. Example applications include discovering stock with similar price movements, identifying companies with similar sales patterns, etc.

There are other data mining activities like classification [Agrawal et.al, 92; Morishita, 98] and clustering [Jain et.al, 99]. **Association Rule Mining (ARM)activity** forms the main focus of this thesis.

1.3 Problem Definition

A database system is designed around the known relationships between sets of values/attributes to structure the data. Such structuring helps in efficient retrieval of data. If specific associations between sets of values are not known, then it is tacitly assumed that these sets of values are independent. However, in the real world, there may exist possible hidden dependencies among values of one or more attributes. Some of these dependencies provide useful results. Data mining brings out these useful hidden dependencies among values.

1.4 Current Status of ARM Activity

Mining a single database for association rules is addressed in the literature. However, most of the ARM activity is on a single transaction database and the association rules are generated using statistical methods. The task here is to find a collection of association rules in the form $A_1 \wedge A_2 \ldots \wedge A_m \Longrightarrow B_1 \wedge B_2 \ldots \wedge B_n$, where each A_i and B_j are values of attributes, from relevant datasets in a database. Associations can be found between values of the *same* or *different* attributes in a relation(s) in the database. For example, one may find, from a large set of transaction data, an association rule - "if a customer buys (brand X) bread, s/he usually buys (brand Y of) butter" - in the same transaction. This is a *positive* association rule. On the other hand, one may find an association rule - "if a customer buys cotton dress, s/he usually does not buy sky shoes". This is called a *negative* association rule. It is possible that A_i and B_j generalized values. In such a case, association rules are called *generalized* association rules. The knowledge in the form of *is-a* hierarchy is used to obtain the generalized values from a set of attributes values. For example, Bajal *is-a* Soft_drink, and from an association rule, Bajal $\Longrightarrow$ Chips, one may infer a generalized rule, Soft_drink $\Longrightarrow$ Chips.

Typically, mining for association rules requires multiple scans of the database [Han et.al, 00]. Also, the amount of data to be processed is huge. So, performance improvement in terms of the computational resources used is the main concern while mining such rules.

Most of the ARM algorithms are data-driven and hence generate a large number of association rules. However, all the rules generated may not be interesting to the user [Silberschartz et.al, 95; Bing Liu et.al, 99; Roberto et.al, 99B].

Activities like addition and deletion of tuples change the status of the database. For efficient extraction of information from changing data, the original database should not be referred to more than *once*, but knowledge derived should reflect the updated database. Such a mining activity is called *incremental mining* [Cheung et.al, 97; Thomas et.al, 00].

From the above discussion, we can make the following observations on the existing ARM activity.

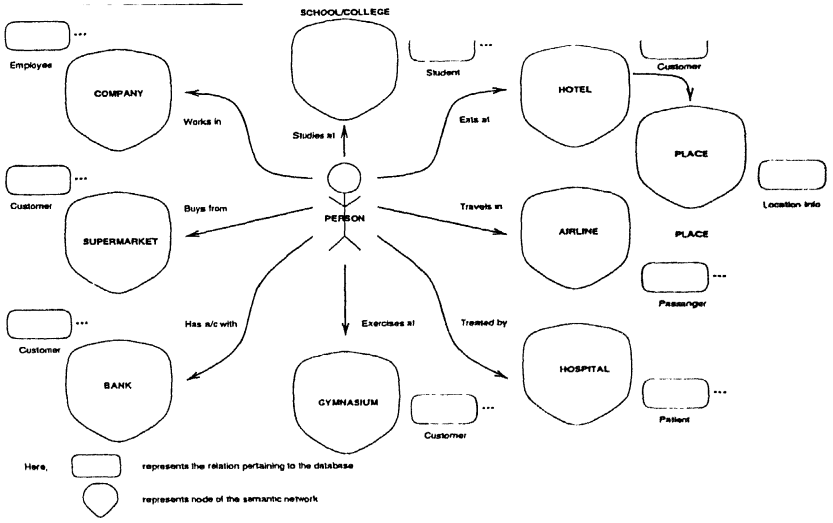
- It is based on a single database; commonly on a transaction database.
- It employs knowledge in the form of *is-a* hierarchy for generalization.
- It is based on co-occurrence of values in a tuple.

In addition, at least *two* scans of the database is required to discover association rules.

1.5 Motivation

In the real world applications, we have a large collection of data that is organized in the form of a set of relations which is partitioned into several databases. Mining relations coming out of different databases might reveal useful hidden associations. For example, the data about a person's behaviour is distributed across several organizations. Figure 1.2 depicts a semantic network [Findler, 79] which shows person's behaviour. Each node in Figure 1.2 can be viewed as an organization, which maintains a collection of databases pertaining to persons. More specifically, CUSTOMER database describes PERSON's service entertainment behaviour; EMPLOYEE database gives his professional information; PATIENT database gives his medical history; and AIRLINE database gives PERSON's travel details. There could be an implicit association among various parts of this data. For example, there may be an association between salary and goods purchased; like *89% of employees having salary in the range US$ 5,000 - US$ 10,000 own Toyota.* There

Figure 1.2 Semantic network



may be an association between salary, communication instrument, mode of traveling and disease; like *people who have salary in the range of US$ 10,000 - US$ 20,000, use cellular phones and travel by air in executive class, have brain tumor.* These examples provide associations between person's behaviour in different contexts. These hidden associations can be discovered if we mine *across* databases.

Currently, most of the association rule mining (ARM) activity is based on mining a transaction (single) database where the attributes may be multi-valued. But the set of values of attributes between which we want to find the associations may be distributed across several relations/databases. So, it is desirable that

- The ARM activity is capable of mining across multiple databases for associations.

Selection of relevant databases for ARM activity is one of the important problems. To mine multiple databases for associations, a brute force approach is to join the related tables corresponding to a collection of databases into a single system, upon which the ARM techniques can be applied. But there are problems in this approach. First, the size of the resultant table may be larger than the input tables. The increase in size not only prolongs the running time of mining algorithms, but also affects the behaviour of mining algorithms. Second, useless or uninteresting patterns are generated as there could be a

number of irrelevant databases encountered while performing join operation. So, it is good to identify the relevant databases and generate abstractions from the selected databases. We use the knowledge in the form of a semantic network to identify the relevant databases and propose schemes to generate intermediate representations of data (i.e., abstractions).

Even though one can generate a vast amount of associations from a collection of databases, in practice only a relevant subset of this knowledge that aids in decision making is of interest. Such a mining activity uses knowledge for mining for only those concepts that are useful for decision making. We use a semantic network to represent the knowledge. This not only reduces the rules generated but also can increase the efficiency of mining in terms of time and space. Typically, most of the current ARM activity involves retrieving of all possible association rules from the data. So, it is desirable that

- The ARM activity is guided by knowledge.

The conventional ARM activity is based on "co-occurrence of values" in *a* tuple of the database. This kind of mining activity is mainly aimed at market-basket analysis and it typically establishes an association between two non-overlapping sets of frequently bought items [Agrawal et.al, 93B]. However, the sets of values of attributes between which we want to find the associations may be distributed across several relations/databases. For example, in finding the associations between the values of person's *age* and *magazine* s/he reads, attributes *age* and *magazine* may not appear in the same relation. Even in such cases, it is possible to come out with interesting and useful associations. As an example, finding the collection of 'magazines' bought by people of a particular 'age' group could be interesting. Note that here, mining for associations is *transaction independent* unlike the conventional association rule mining activity, because a person may not buy all the magazines together. Here, the idea is - the values need not co-occur in the same tuple; however, they may be associated with each other. We call such an association, *indirect co-occurrence*. So,

- The ARM activity for indirect co-occurrence of values is desirable.

In real life, databases are dynamic. Along with efficient handling of changing data incrementally, (i.e., incremental mining), the other requirements for mining in a dynamic

environment are to handle change of knowledge and change of user defined support value. Handling of change of knowledge is required because knowledge of several domains keeps changing with time; for example, the Dewey-decimal classification system [George, 91] employs dynamic knowledge. Most of the times, in the mining activity for finding frequent patterns, the support value is fixed and is chosen arbitrarily. This is because user may have to experiment with different support values. So, there is a need for changing support value for mining frequent patterns. We call the scenario of mining under - *change of data*, *change of knowledge* and *change of support value* without having access to the database for every change - *dynamic mining* [Ananthanarayana, 00B]. Current ARM activity can handle only incremental mining. However, it is beneficial if

- The ARM activity is capable of providing *dynamic mining* environment.

It is quite natural in the real life that the databases are distributed but have the same schema. This may be due to the geographically distributed branches of a firm, where each branch maintains the database corresponding to that locality. Activity of collectively generating association rules based on distributed data is called *distributed mining*. So, it can be efficient if

- The ARM activity is capable of handling *distributed mining*.

Mining of association rules requires intensive disk activity in terms of scanning the database many number of times. Since, the disk access is a milli-second operation whereas a memory access is a nano-second operation, it is important to *reduce the number of database scans* and hence the number of disk accesses. So, an important feature of an efficient mining algorithm is,

- It should use a small number of database scans; preferably a single scan of the database.

Most of the ARM algorithms available in the literature are aimed at reducing the number of database scans. The best possible algorithm [Han et.al, 00] requires two database scans.

It is observed in most of the association rule mining algorithms that they generate the patterns of length $(k+1)$ iteratively, without looking at the data, from set of frequent patterns of length k (for $k > 1$). Such iteratively generated patterns are called *candidate*

patterns. This candidate generation process not only blows-up the memory requirement to store candidate patterns; but also increases the processing time. So, the desired feature of an ARM algorithm is that

- It does not generate any *candidates*.

Typically, a mining algorithm generates an abstraction of data in the databases for its local consumption. For the better use of resources, size of such abstractions generated and used by the algorithm should be small. So, it is desirable that,

- Abstraction generated by the ARM algorithm is *compact*.

1.6 Scope of the Work

The needs of ARM activity and features of ARM algorithms which are discussed above have motivated us to carry out the work reported in this thesis. More specifically, we address the following issues :

- Compact abstraction generation

  In this thesis, we develop schemes where the abstraction generated for the local consumption of the algorithm is *small* in size.

- Single scan of the database

  The best possible algorithm reported in the literature requires at least two database scans [Han et.al, 00]. In this thesis, we propose and use novel schemes which require *a single scan* of the database for finding associations.

- No candidate generation

  In this thesis we develop schemes where there is no candidate generation.

- Multiple database-oriented ARM activity

  Mining multiple databases for associations is not addressed in the literature [Subramanian et.al, 99]. We call such associations, *inter-database associations*. In

this thesis, we propose and implement efficient schemes to obtain inter-database associations. The proposed schemes use domain knowledge in the form of a *semantic network* to link various databases.

- Semantic modeling

  In ARM activity, knowledge in the form of *is-a* hierarchy is used for discovering generalized rules. In this thesis we propose a taxonomy, a kind of semantic network based on functionality and a restricted way of generalization of the features. We claim that this type of generalization is more meaningful since it is based on a semantic-grouping of concepts. We used that knowledge for *relevant*, *interactive* association rule mining activity. We also use the knowledge in the form of semantic network to identify the relevant databases for mining of multiple databases.

- Dynamic mining

  ARM activity based on change of data, called incremental mining [Cheung et.al, 96A; Thomas et.al, 97], and change of support value [Han et.al, 95] are described in the literature. Handling of *change of knowledge* without having access to the original database is not reported in the literature. In this thesis, we propose novel schemes which can handle all possible facets of *dynamic mining*.

1.7 Thesis Contributions

This thesis focuses on key techniques in association rule mining activity in view of the scope of the work discussed above.

A review and survey of association rules is given in Chapter 2. In this chapter, we give survey of different types of - association rules and association rule mining algorithms. We also give a general structure for association rule mining algorithms.

The notations and definitions used in this thesis, problem formulation, solution methodology for association discovery proposed and used by us are discussed in Chapter 3.

The research work presented in this thesis covers a spectrum of novel methods and

data structures for efficient generation of associations. In Chapter 4, we discuss the generation of association rules across multiple databases. Domain knowledge in the form of semantic network is used to identify the relevant databases. We generate navigation path based on schema integration methodology. We propose and implement a new structure called Extended Inverted File (EIF) to hold associations across the databases and it semantically partition the resultant database. In Chapter 5, we propose a new knowledge based tree, called And-Or taxonomy (AO taxonomy). We discuss ARM activity based on such a structure along with implementation. In Chapter 6, we propose and implement a collection of novel schemes based on Pattern Count Tree (PC-tree). In Chapter 7, we propose a new kind of association rule called 'Dominant Entity Based Association Rule (DEBAR)'. DEBAR is different from the conventional association rule in the sense that it is based on co-occurrence of values that exist in different tuples. We propose and implement a novel structure called Multi-database Domain Link Network (MDLN) which can be used to generate DEBARs between the values of attributes belonging to different databases. Finally, Chapter 8 provides discussion and summarizes the main contributions of this thesis, and suggests avenues for future work.

Chapter 2

Literature Survey and Review

2.1 Introduction

In this chapter, we review different types of association rules described so far in the literature. A general scheme to represent the process of mining association rules with the problem statement is given in section 2.2. Survey on relevant association rules are discussed in section 2.3. Section 2.4 gives a detailed description of association rules based on their structural and semantic properties. In section 2.5, we undertake classification of association rule mining (ARM) algorithms. In section 2.6, we present abstractions of categories of algorithms based on structural and semantic properties. We also discuss the characteristic of each category of abstraction. In section 2.7, we discuss the general structure of algorithms that generate frequent itemsets. Section 2.8 summarizes this chapter.

2.2 Mining Association Rules

Data mining is a step in the process of Knowledge Discovery in Databases (KDD) and is based on the use of particular mining algorithms which produce patterns that are non-trivial, previously unknown but potentially useful. Various data mining techniques are available - to improve the business opportunities, to understand user behaviour and the

service provided in a better manner. Survey of various data mining techniques are made in [Chen et.al, 96] and [Agrawal et.al, 93A]. They are typically based on :

1. Categories of discovered knowledge : Such as association rules, sequential patterns, class descriptions, clustering, time sequences, data-generalization, summarization and characterization.

2. Techniques utilized : Such as data-driven, knowledge-driven, query-driven and interaction-driven.

Mining association rules in transactional or relational databases has recently attracted the attention of the database community. The task is to derive important associations among items such that *presence* or *absence* of some items in a transaction will imply the *presence* or *absence* of some other items in the same transaction.

For example, 'Alsatian_dog $\Longrightarrow$ brand_a_biscuits' is an association rule which shows the existence of association between Alsatian_dog and brand_a_biscuits. Another example, 'asthma_patient $\not\Longrightarrow$ watermelon' means asthma_patient and watermelon are negatively associated. The food of asthma patient should not contain watermelon. Here the aim is to find associations between collection of items based on their frequency of co-occurrences. Mining for association rules consists of determining the correlation between items in transactions belonging to a transaction database [Aggrawal et.al, 97;Agrawal et.al, 93A; Hidber, 99; Agrawal et.al, 93B; Han et.al, 00; Agrawal et.al, 94; Savasere et.al, 95; Agrawal and Srikant, 95; Zaki et.al, 97; Agrawal and Shaffer, 96; Chen et. al, 96; Holsheimer et.al, 95].

2.2.1 Problem Statement

Let I be a set of items, and D a database of transactions, where each transaction has a unique identifier (*tid*) and contains a set of items. A set of items is also called an *itemset*. An itemset with k items is called a k-itemset. The *support* of an itemset X, **Support**(X), is the ratio of number of transactions in which it occurs as a subset to the total number of transactions in the database. An itemset is *frequent (large)* if its support is more than a user specified *minimum support* (σ) value.

An *association rule* is an expression of the form $\mathfrak{A} \Longrightarrow \mathfrak{C}$, where $\mathfrak{A}$ and $\mathfrak{C}$ are set of items. Left hand side of the rule ($\mathfrak{A}$) is called the antecedent and the right hand side of the rule ($\mathfrak{C}$) is called the consequent. In literature, a constraint $\mathfrak{A} \cap \mathfrak{C} = \emptyset$ is tacitly assumed. Every association rule must satisfy two user specified constraints, one is *support* and the other is *confidence*. The support of the rule $\mathfrak{A} \Longrightarrow \mathfrak{C}$ is defined as the fraction of tuples that contain both $\mathfrak{A}$ and $\mathfrak{C}$. In other words, it is the **Support**($\mathfrak{A} \cup \mathfrak{C}$). Confidence of the rule is defined as **Support**($\mathfrak{A} \cup \mathfrak{C}$) / **Support**($\mathfrak{A}$). The goal is to find all rules that satisfy minimum support and minimum confidence specified by the user. Note that the implication ($\Longrightarrow$) is not a logical implication. This is because *for every* presence of $\mathfrak{A}$, $\mathfrak{C}$ may not present in the *same* tuple; but they present together by a fraction '**Support**($\mathfrak{A} \cup \mathfrak{C}$)'.

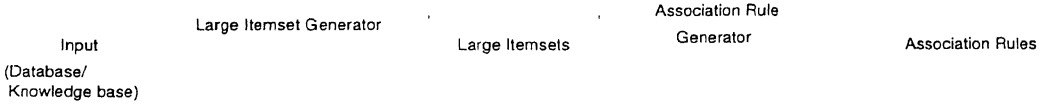
Mining association rules consists of two parts.

1. Discovery of all *frequent (large) itemsets.* i.e., the set of items that have support greater than or equal to a user defined minimum support (σ).

2. Use the frequent itemsets to generate association rules using the user defined confidence factor (c). i.e., constructing rules of the form $\mathfrak{A} \Longrightarrow \mathfrak{C}$ from the frequent itemsets, such that ratio of number of tuples having itemsets $\mathfrak{A}$ and $\mathfrak{C}$ together to the number of tuples having itemsets $\mathfrak{A}$ in the transaction database is greater than or equal to c.

The process of generating association rules is depicted in block diagram 2.1. In this

Figure 2.1 A general block diagram to represent the process of mining association rules



diagram, input to the process is either Database or Knowledge base or both. Depending on the input, the process of generating association rules is called as **Data driven** or **Knowledge driven**.

2.3 Relevant Association Rules

All the association rules generated by the algorithms may not be useful. Most of the algorithms reported in literature are designed to generate a complete set of association rules which are possible based on the user specified support and confidence. These algorithms have limitations; due to generation of possibly a huge number of frequent itemsets it is very difficult to focus on the right patterns which are interesting to the user; further the computational requirements of the algorithms also increase. This motivated the researchers in this area to design algorithms for generating 'relevant/interesting' association rules. Many measures of interestingness have been proposed. They are classified as *objective measures* and *subjective measures* [Silberschartz et.al, 95; Bing et.al, 99]. Objective measures are those that depend only on the structure of the pattern. Measures like confidence, strength [Dhar et.al, 93], performance [Major et. al, 93], simplicity [Major et.al, 93], statistical significance [Major et.al, 93], gain [Fakuda et.al, 96A], gini [Morimoto et.al, 98], laplace [Clark et.al, 91; Webb 95], variance, chi-squared value [Morishita, 98], entropy gain [Morimoto et.al, 98], lift [Roberto et.al, 99B], conviction [Brin et.al, 97; Roberto et.al, 99B] are objective measures. On the other hand, subjective measures depend on the requirements of the users who examine the pattern. Measures like unexpectedness and actionability [Silberschartz et.al, 95] constitute subjective measures. Filtering of transactions based on criteria like cost of transactions [Raymond et.al, 98], grouping of items/concepts [Han et.al, 95] belong to this category.

2.4 Characterization of Association Rules

Association rules are characterized based on *structure*, *semantics* and *hybrid* of both structure and semantics. We discuss each issue below.

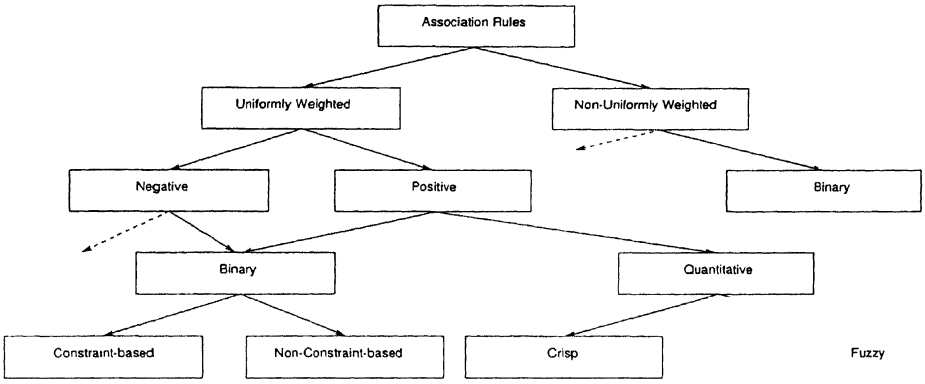
2.4.1 Based on Structure of Association Rules

The structure of an association rule was very simple, i.e., in the form of a relationship between itemsets, in the early proposals. However, an active research in this area over the

recent years has lead to a variety of structures to characterize association rules. In this subsection, we summarize various structural forms by providing a classification based on commonly used abstractions.

Figure 2.2 shows the classification of association rules based on structural properties. The dashed line for an edge in figure 2.2 indicates that in the literature there are no

Figure 2.2 Classification of association rules



instantiations of rules of classes corresponding to the subtree below that edge.

Based on the presence or absence of itemsets in the formation of the association rule, we have **Positive association rules** and **Negative association rules** [Savesere et.al, 98].

Positive Association rules : Positive association rules are of the form $X \Longrightarrow Y$ where X and Y are itemsets and the presence of X implies presence of Y. The intuitive meaning of the rule is that the transactions of the database which contain X tend to contain Y. For example, we may find that "90% of the customers who buy bread also buy jam in the same transaction". Here items bread and jam are positively associated [Agrawal et.al, 93A; Agrawal et.al, 93B].

Negative Association rules : They are the complement of the above rules, where presence of X implies the absence of item Y when compared with minimum support of Y alone. Such rules are represented as $X \not\Longrightarrow Y$. For example, "60% of the transactions in which potato chips are brought, bottled water is not bought" [Savasere et.al, 98].

Based on the items and itemsets used in the formation of the association rules, we

have **Binary association rules** and **Quantitative association rules** [Srikant et.al, 96; Fakuda et.al, 96A; Fakuda et.al, 96B; Han et.al, 93].

Binary Association rules : Here each transaction t is represented as a binary vector, with $t[k] = 1$, if item k is bought in t and $t[k] = 0$ otherwise. Let X be a set of some items in I. We say that a transaction t satisfies X if for all items i_k in X, $t[k] = 1$. [Agrawal et.al, 93B].

Quantitative Association rules : Tables in most business and scientific domains have richer attribute type. Attribute can be quantitative (eg. age, income) or category-based (eg. zip code, make_of_car). The problem is to identify association rules between quantitative and category-based data in relational tables. A quantitative association rule is of the form $(Age : 30..39) \wedge (Married : Yes) \Longrightarrow (No.ofCars : 2)$. [Srikant et.al, 96; Fakuda et.al, 96A; Fakuda et.al, 96B; Han et.al, 93].

Based on variations in support value in the formation of the association rules, we have **Crisp association rules** and **Fuzzy association rules** [Kuok et.al, 98].

Fuzzy association rules : These are the fine-tuned versions of quantitative association rules [Srikant et.al, 96], where fuzzy sets are used. In [Srikant et.al, 96], attribute domains are partitioned and mapped to a binary value. Because of this we end up with the sharp boundary problem and hence we miss some of the interesting intervals at sharp boundaries. On the other hand, we can use fuzzy association rules of the form : "if X is A then Y is B; (X is $A \Longrightarrow Y$ is B)" [Kuok et.al, 98], which provides a smooth boundary. Here, each attribute x_k in X will have a fuzzy set f_{xk} in A such that f_{xk} belongs to F_{xk}, a set of all fuzzy sets, and is similar for attribute Y. For example, during 'peak summer' days consumption of cool drinks is 'high'.

Based on provision for the specification of constraints in the formation of the association rules, we have **Constraint-based association rules** [Srikant et.al, 97; Roberto et.al, 99; Thomas et.al, 00] and **Un-constrained association rules.**

Constraint based association rules : These rules deal with discovering association rules by applying different constraints to different part of the data [Srikant et.al, 97; Roberto et.al 99]. Constraints are broadly classified into five categories [Han et.al, 99].

They are *knowledge constraints* - specifies type of knowledge, i.e, association, clustering, classification, etc to be mined; *data constraints* - specifies relevant data to be mined using queries; *dimension/level constraints* - specifies levels of data to be mined; *rule constraints* - mining based on rule specifications; *interestingness constraints* - based on important concept/items to be included in the rule, ranges of measures, etc.

Based on assignment of weightage to different terms in a frequent itemset, we have **Uniformly weighted association rules** and **Non-uniformly weighted association rules** [Cai et.al, 98].

Weighted association rules : In normal way of mining association rules, uniform weight is given to all items in a supermarket database. But due to a business strategy, more emphasis could be given to some items and less emphasis is given to other items. For example, the rule, $Coffee\_Powder \Longrightarrow Car$, which is more unexpected, is more interesting than $Coffee\_Powder \Longrightarrow Milk$. So, depending on 'interestingness' of the items, different weights could be given to each of the items in the supermarket. The association rules generated using this criteria are called weighted association rules [Cai et.al, 98]. Here, a k-itemset is a large itemset if its *weighted support* is greater than or equal to user specified minimum support. The weighted support of an itemset is given by summation of weights of each item in that set, multiplied by the support of that itemset.

2.4.2 Based on Semantics of Association Rules

We have two categories under this characterization. They are based on

Location of attributes under consideration :

The association rule is of the form $\mathfrak{A} \Longrightarrow \mathfrak{C}$. Depending on the attributes from where set of values, $\mathfrak{A}$ and $\mathfrak{C}$ are selected, we have following *four* categories. (i) $\mathfrak{A}$ and $\mathfrak{C}$ belong to an attribute in a relation (ii) $\mathfrak{A}$ and $\mathfrak{C}$ belong to different attributes in a relation (iii) $\mathfrak{A}$ and $\mathfrak{C}$ belong to attributes of different relations in a database (iv) $\mathfrak{A}$ and $\mathfrak{C}$ belong to attributes of relations of different databases.

Most of the work reported in literatures on 'mining for association rules' are belong to the category (i); literature dealing with work on category (ii) and (iii) are scanty;

viz., [Han et.al, 92; Luc Dehaspe et.al, 97; Knobbe et.al, 99]. Category (iv) is not reported in the literature from the best of our knowledge. We explore category (iv) in this thesis.

Knowledge Structure

In an association rule, both or any of antecedent and consequent may be the categorical value based on knowledge structure. The knowledge in the form of *is-a* hierarchy is used to generate association rules, called generalized association rules [Srikant et.al, 95] and multiple-level association rules [Han et.al, 95]. Little work is done under this category. We explore this category further in this thesis.

2.4.3 Based on Both Structure and Semantics of Association Rules

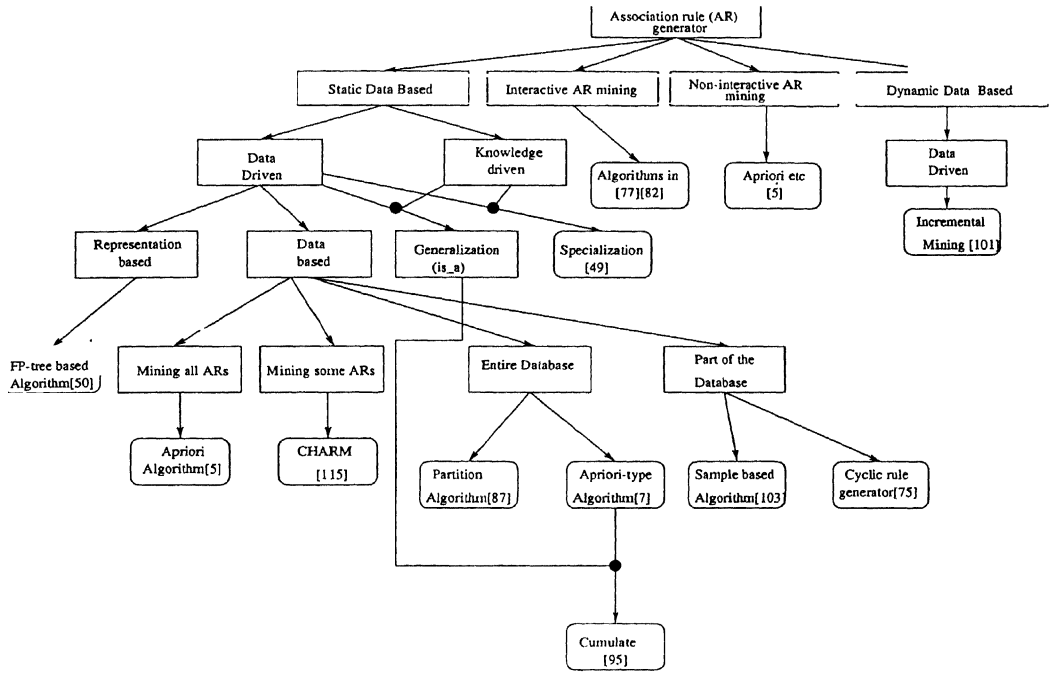
The association rules discussed in [Savasere et.al, 98] belong to this category. For example, consider that *Bajal is-a Soft-drink* and *Soft-drink* $\Longrightarrow$ *Chips*. However, *Bajal* $\not\Longrightarrow$ *Chips* is a negative association rule. This rule has different structure, i.e., $v_i \not\Longrightarrow v_j$ and uses knowledge in the form of *is-a* hierarchy.

2.5 Classification of Algorithms for Mining Association Rules

There is a large collection of ARM algorithms in the literature and the size of this collection is increasing day by day. Such a vast collection might lead to difficulties in understanding the behaviour of the algorithms in a comparative manner. This motivated us to structure the collection based on the type of input provided to the algorithm. Such a structuring helps in abstracting the behaviour of algorithms and as a consequence in carrying out a comparative study.

The collection of ARM algorithms can be represented in an abstract form using a tree structure as shown in Figure 2.3. Note that each leaf node of the tree in Figure 2.3 is

Figure 2.3 Classification tree for ARM algorithms



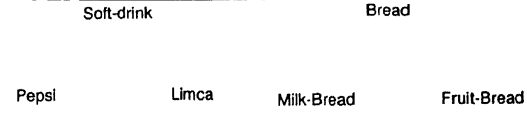
pointing to an ARM algorithm which is appropriate to generate frequent itemsets for the classification from root leading to that leaf node. The structure shown in Figure 2.3 is obtained by agglomerating various dichotomies that result based on variation in the input to the algorithm. These dichotomies are considered below.

- **Knowledge driven Vs. Data driven**

  ARM algorithms based *only* on data in the database (data-driven) can generate a large number of association rules. In such cases, use of a knowledge base to filter out trivial rules may reduce the effort of the user to mine for interesting rules and also reduce the number of database scans. Knowledge driven algorithms generate either generalized itemsets or high level concepts [Srikant et.al, 95].

  For example, let us consider knowledge in the form of an "is_a" hierarchy shown in Figure 2.4. In the figure, Pepsi is_a soft drink, Limca is_a soft drink, Milk_bread is_a bread, etc. Using this taxonomy of information, rules can be generated which span different levels of the taxonomy. For example "72% of customers who buy *soft*

Figure 2.4 An example is_a hierarchy



drinks tend to buy *bread*". Rules of this form are called **generalized association rules.**

- **Entire database based Vs. Part of the database based**

  This classification is based on whether the entire database is used for mining large itemsets or mining is based on a part of the database. The algorithms in [Agrawal et.al, 93B; Park et.al, 95; Sarawagi et.al 98; Tsur et.al, 98; Agrawal et.al, 94; Agrawal et.al, 95A; Savasere et.al, 95; Raman et.al, 98; Cheung et.al, 96A; Thomas et.al, 97] use the entire database, where as the algorithms reported in [Toivonen, 96; Zaki et.al, 97A] use samples for constructing candidate itemsets.

  If an algorithm scans a part of the database corresponding to a time interval to generate the association rules, the rules generated are called **Cyclic association rules** [Ozden et.al, 98], otherwise they are called **Acyclic association rules.** For example, in a database maintained by a cafeteria, an association rule might be of the form 'coffee $\Longrightarrow$ toast' (support =3% , confidence =87%). But it may be the case that coffee and toast are sold together primarily between 8AM and 10AM. Therefore, if we segment the data over the intervals 8AM - 5PM such that each segment carries transactions pertaining to two hour durations, we may find that the support for the rule, coffee $\Longrightarrow$ toast, jumps to 60% during 8AM-10AM segment, against 3% during 8AM-5PM. From this example, we can conclude that although an association rule may have the user specified minimum confidence and support within the entire time spectrum, analysis of the data in a fine-time granularity may reveal that the association rule exists only in certain time intervals and does not get supported in the remaining time intervals.

- **Mining all association rules Vs. Mining some association rules**

  This classification of algorithms is based on whether the algorithms mine all or some of the association rules.

  Algorithms reported in [Zaki et.al, 99A; Pai et.al, 00] do not generate all association rules. The basic idea behind these algorithms is the generation of *frequent closed itemsets*. Such association rules are also called **closed association rules**. An itemset I is a *closed itemset* if there exists no itemset $\acute{I}$ such that $\acute{I}$ is a proper superset of I and every transaction containing I also contains $\acute{I}$. If support of closed itemset, I is greater then or equal to user specified support threshold, then I is a frequent closed itemset. Thus, instead of mining association rules on all frequent itemsets, one can mine association rules on frequent closed itemsets. Such association rules are called closed association rules. Since number of frequent closed itemsets is smaller than the number of frequent itemsets and association rules based on closed itemsets contain the association rules based on frequent itemsets, no redundant rules are generated.

- **Database based (Direct) Vs. Abstraction based**

  A majority of ARM algorithms directly work on the database by employing multiple database scans. However, it is a good idea to represent the database in a compact form to reduce the number of database scans for mining for large itemsets. Such representations are discussed in [Han et.al, 00]. The main issue in using these representations is to reduce the number of database scans by compactly representing the database in a suitably complete form.

- **Static mining Vs. Incremental mining**

  In this case, the classification of algorithms is based on whether they generate frequent itemsets based on changing data or not. Algorithms that can handle such changing scenario are called **incremental mining algorithms**. Algorithms that cannot handle such changes are called **static mining algorithms**. Most of the

algorithms reported in the literature are static mining algorithms. Algorithms discussed in [Cheung et.al, 96A; Thomas et.al, 97; Pudi et.al, 00; Cheung et.al, 98; Sarda et.al, 98] are example for incremental algorithms.

- **Interactive mining Vs. Non-interactive mining**

Most of the association rule generating algorithms perform in auto mode, since there is no interaction of user, once s/he supplies thresholds at the beginning, and algorithms output all association rules at the end. Such algorithms are called "non-interactive mining algorithms". On other hand, interactive mining algorithms allow the user to specify the types of rules s/he is interested in - like - identify the k most important rules using given support, confidence pairs; find out for what support/confidence values one can generate exactly k rules; find the rules - which have set of items of his interest, or which do not have some set of items; generate rules which have user specified items or user specified number of items in antecedent/consequent parts of the rules. Interactive mining algorithms are reported in [Parthasarathy et.al, 99; Raymond et.al, 98].

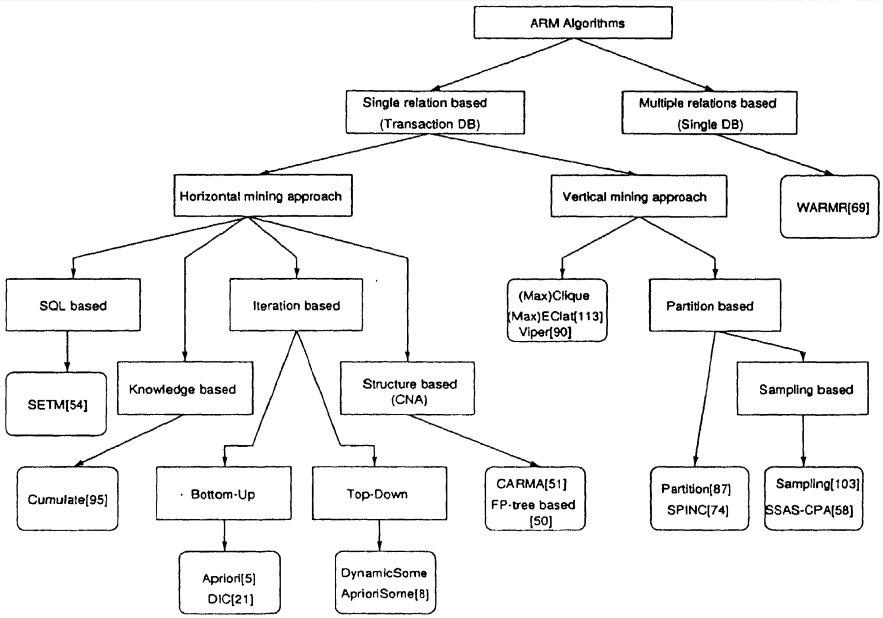
- **Sequential Vs. Parallel mining**

Depending on whether the mining algorithms are based on sequential or parallel processing of database, one can classify the algorithms as sequential mining or parallel mining algorithms. The work reported in [Eui-Hong et.al, 97; Agrawal et.al, 96; Shintani et.al, 98] deals with parallel algorithms. Mining on distributed databases using a distributed mining algorithm is discussed in [Cheung et.al, 96B]. Scale-up, size-up, speed-up, response time, efficiency of parallelism are the main parameters based on which the performance of these algorithms is studied and parallel algorithms are compared against their sequential counterparts. A detailed survey on parallel and distributed association rule mining is given in [Zaki et.al, 99B].

2.6 Abstraction of ARM Algorithms

In this section, we categorize ARM algorithms based on their structure and present an abstract version of each category. Taxonomy of ARM algorithms is shown in Figure 2.5.

Figure 2.5 Taxonomy of ARM algorithms



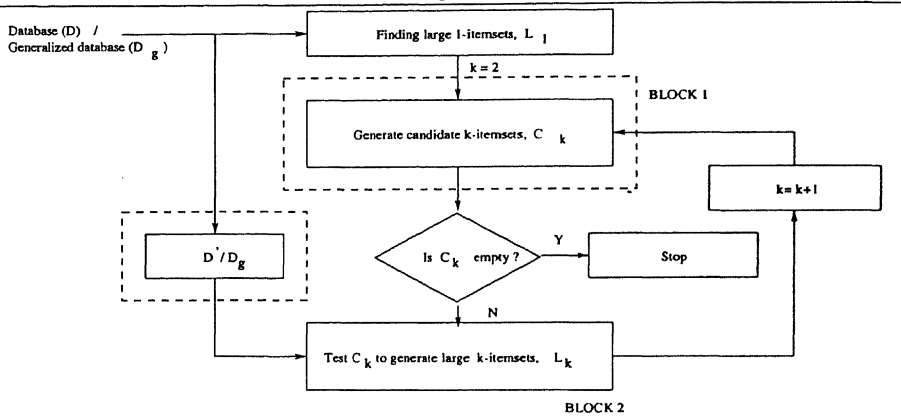
2.6.1 Iteration Based Algorithms

These are algorithms where the candidate k-itemsets are generated from large (k-1)-itemsets in a level-wise manner. Algorithms - **Apriori** [Agrawal et.al, 93B], **AprioriTid** [Agrawal et.al,94], **DHP** [Park et.al, 95], **DIC** [Brin et.al, 97], **FITI** [Tung et.al, 99], **FUP** [Cheung et.al, 96A], **FUP$_2$** [David et.al, 97], **AprioriAll / AprioriSome / DynamicSome** [Agrawal et.al, 95A], **(Max)Eclat, (Max)Clique** [Zaki et.al, 97B], **Quantitative association rules** [Srikant et.al, 96], **Dense Miner** [Roberto et.al, 99A], **MINWAL** [Cai et.al, 98], **ML-T1LA** [Han et.al, 95], **Negative association rules** [Savasere et.al, 98] and **Cumulate** [Srikant et.al, 95] - belong to this category.

2.6.1.1 An Abstract Version of Iteration Based Algorithms

Figure 2.6 shows an abstraction of iteration-based algorithms.

Figure 2.6 Abstraction of iteration based algorithms



Note that the dashed box in Figure 2.6 is the variable part among the algorithms belonging to this category. BLOCK 1 in Figure 2.6 is called **generation phase**; BLOCK 2 is called **test phase**. In test phase, the database is scanned to see whether the candidate itemsets are large or not based on the user defined minimum support, σ.

Now we consider the individual algorithms under this category.

Basic Algorithms

Apriori : In this case, all candidate itemsets (C_j) are generated at the beginning of the iteration. Here $\acute{D} = D$.

Apriori_Tid : In this case, $\acute{D} < D$. The database size to be scanned is reduced in each iteration i by discarding transactions which have no itemsets of size $(i-1)$ having support $\geq \sigma$ corresponding to itemsets in L_{i-1}.

Direct Hashing and Pruning (DHP) : When support of candidate k-itemsets (C_k) is counted by scanning the database, DHP algorithm accumulates the information about candidate $(k+1)$ itemsets in advance in such a way that all possible $(k+1)$-itemsets of each transaction are hashed to hash table. Each entry in the hash table

is a counter that maintains the sum of the supports of the $(k+1)$-itemsets. The algorithm uses this information in iteration $(k+1)$ to prune set of $(k+1)$-itemsets. This idea reduces number of candidate itemsets in BLOCK 1. Like Apriori_Tid, it also reduces the database size (i.e, $\acute{D} < D$) by eliminating transactions that do not contain large k-itemsets while determining set of large $(k+1)$-itemsets.

Dynamic Itemset Counting (DIC) : This algorithm does not generate all candidate itemsets at the beginning of the iteration like the ones discussed above. Instead, it adds new candidate itemsets at any point, tp_i, of database scan. At each tp_i, DIC estimates the support of all itemsets that are currently counted and adds new itemsets to the set of candidate itemsets if all its subsets are estimated to be frequent. Note that in this case, $\acute{D} = D$.

Variations :

First Intra Then Inter (FITI) : This is an algorithm for finding inter-transaction association rules. It generates intra-transaction itemsets using Apriori algorithm during Phase 1 and generates a data structure called Frequent Itemsets Linked Table (FILT). In Phase 2, it generates inter-transaction itemsets.

Fast UPdate (FUP) : This is an algorithm to compute large itemsets when the database (DB) is updated by adding new transactions (db). It reuses the existing large itemsets of DB and generates candidate itemsets from db. But basic framework of FUP is similar to that of Apriori.

Fast Update$_2$ (FUP$_2$) : This is an extended version of FUP. It not only handles addition of new transactions, but also handles deletion from the existing database. Similar to FUP, here also the basic framework is Apriori.

AprioriAll/AprioriSome/DynamicSome : These algorithms generate large sequences as the output. The first algorithm (AprioriAll) counts all large sequences including non-maximal sequences. On the other hand, AprioriSome and DynamicSome start with large sequences to avoid generating subset sequences. The difference

between AprioriSome and DynamicSome is that the former generates candidate sequences from large sequences found in previous pass, where as the latter generates candidate sequences on-the-fly using previously found large sequences.

(Max)Eclat and (Max)Clique : These methods use the vertical layout of transaction database, where each item is associated with transaction ids, *tid-list* where they belong to. Here support of a k-itemset is determined by intersecting *tid-lists* of any two of its (k-1)-itemsets. Since, the intermediate tid-list may not fit into the main memory, the search space is divided into smaller, independent partitions which can be processed in main memory via prefix-based or clique-based classes. Algorithm (Max)Eclat uses prefix-based classes and algorithm (Max)Clique uses clique-based classes.

Algorithm for generating quantitative association rules : It maps quantitative attributes (like Age, Salary) and category-based attributes (like zip code) to the set of consecutive integers. It then uses Apriori algorithm to generate association rules using the mapping information.

Dense Miner : It is a constraint-based mining algorithm which generates all association rules with user defined consequent C. The algorithm uses an iterative procedure to generate set-enumeration tree. During each iteration, it expands the set of items and extracts rules out of this.

Algorithm for mining weighted association rules (MINWAL) : This algorithm adopts the existing Apriori algorithm. But instead of using user defined minimum support directly, it uses weighted minimum support for finding frequent itemsets.

Algorithm for mining multi level association rules (ML-T1LA) : It generates multiple level frequent itemsets using an 'is_a' hierarchy. It is an iterative algorithm, where in each iteration it generates frequent k-itemsets at the k^{th} level of the hierarchy.

Cumulate : It uses Apriori algorithm with an 'is_a' hierarchy to generate generalized association rules. Here the database used is D_g, a generalized version of D.

Algorithm for mining negative association rules : It first finds all generalized itemsets as in Cumulate. Next it identifies the candidate negative itemsets based on large itemsets and the 'is_a' hierarchy. Finally, it counts actual support for candidate negative itemsets to generate only frequent negative itemsets.

2.6.2 Algorithms Based on Partitioning of the Database

These are algorithms where the database is partitioned. Algorithms - **Partition based algorithm** [Savasere et.al, 95], **SPINC** [Mueller et.al, 95] and **AS-CPA** [Jun-Lin et.al, 98] - belong to this category.

Some of the partition based algorithms also use sampling to find association rules. Algorithms - **Sampling Algorithm** [Toivonen, 96], **SSAS-CPA** [Jun-Lin et.al, 98] - belong to this category. Evaluation of sampling techniques for association rule mining is discussed in [Zaki et.al, 97A; Kivinen et.al, 94].

2.6.2.1 An Abstract Version of Algorithms Based on Partitioning of the Database

Figure 2.7 shows an abstraction of algorithms which mine by partitioning the database.

In Figure 2.7, output from PHASE 1 is a set of locally large itemsets, C_g.

Note that the dashed box in Figure 2.7 is the variant part among the algorithms belonging to this category.

Figure 2.8A shows the details of PHASE 1 block of Figure 2.7 for partition-based algorithm and SPINC algorithm. Dashed lines and boxes in Figure 2.8A indicate the control flow and extra processing blocks required by the SPINC algorithm. .

Figure 2.8B shows the details of PHASE 1 for sample based algorithm.

Figure 2.7 Abstraction of algorithms based on partitioning of the database

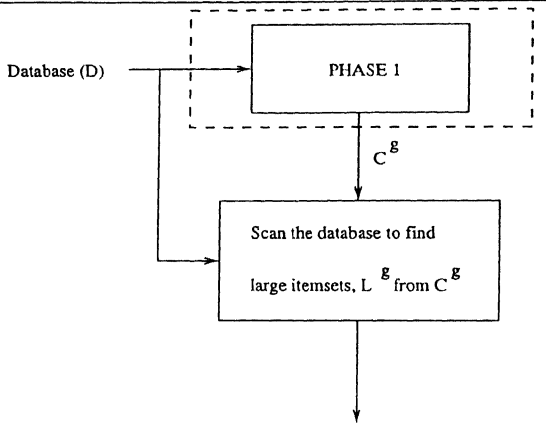
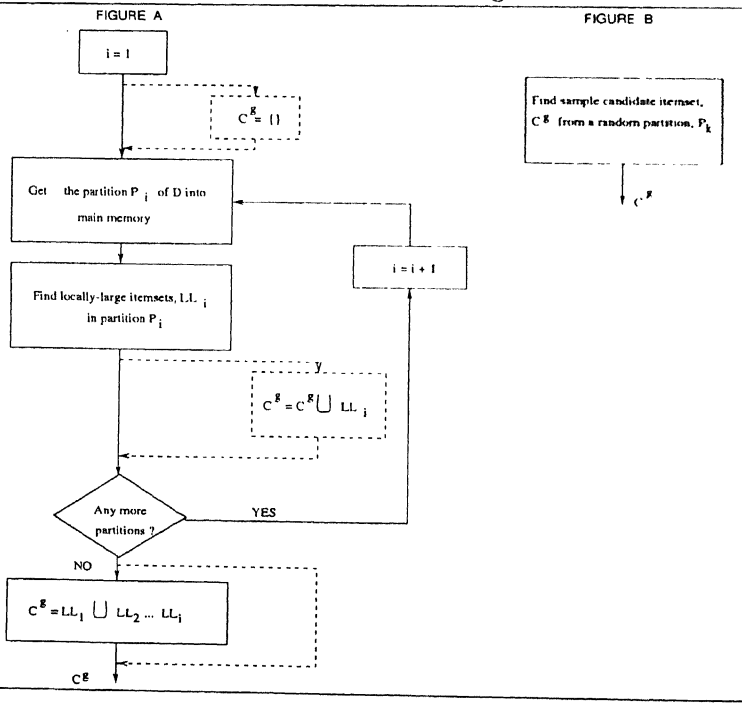


Figure 2.8 PHASE 1 for partition based and SPINC algorithm



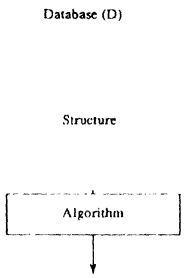
2.6.3 Candidate Non-generation Algorithms (CNA)

These algorithms use some structures internally to find frequent itemsets. The basic idea behind these algorithms is that they do not generate candidate itemsets. They only *test* candidate itemsets, for possible largeness, thus reducing the computational requirements significantly. Algorithms - **CARMA** [Hidber, 99] and **FP-tree based algorithm** [Han et.al, 00] - belong to this category.

2.6.3.1 An Abstract Version of CNA

Figure 2.9 shows the general sketch for structure based algorithms.

Figure 2.9 Abstraction of structure based algorithms

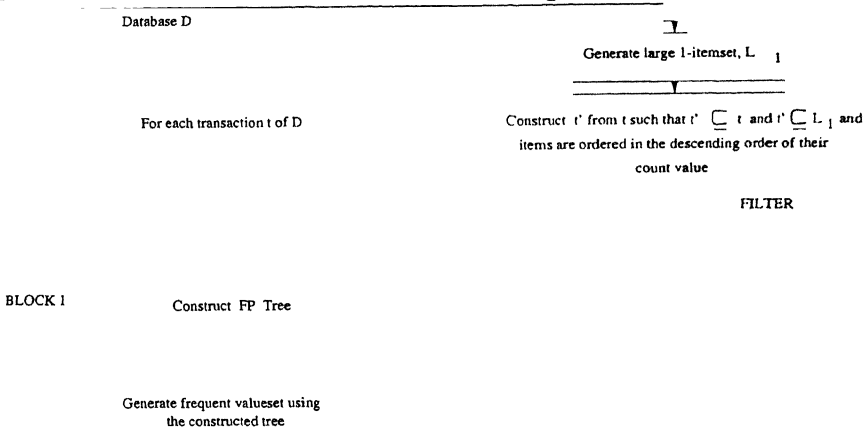


Continuous Association Rule Mining Algorithm (CARMA) : It uses two scans to generate all large itemsets. The structure generated during the preprocessor step is a lattice. During the first database scan, it generates a lattice of all potentially large itemsets with respect to the scanned part of transactions. For each set in the lattice, CARMA provides a deterministic lower and upper bound for its support. In the second scan, CARMA determines precise support of each set in the lattice and continuously removes all small itemsets.

Tree structure based algorithms : Figure 2.10 shows abstraction of FP-tree based algorithm.

Construction of FP-tree (BLOCK 1 in Figure 2.10) is done by treating each transaction (after passing through the FILTER block), as a pattern and repetition of a

Figure 2.10 Abstraction of tree structure based algorithms



part of the pattern, Pi, from the root is handled by incrementing the count field associated with each node along Pi. If the first element in a transaction is different from the item in the first node in any pattern starting from the root, then a new pattern is attached to the root with nodes, each of whose count field is set to 1 for each element in the transaction.

2.6.4 Multiple Relations of Single-database Based Algorithms

Algorithm **WARMR** [Luc Dehaspe et.al, 97] and its variation [Knobbe et.al, 99] discuss generation of association rules over multiple relations belonging to the *same* database.

WARMR Algorithm : It is an extension of the Apriori algorithm to mine associations in multiple relations in a database D. In this algorithm, it is assumed that all relations in D have exactly one common attribute called *Example key (ExKey)*. Candidate itemsets (atomsets) across the relations are generated using *ExKey*. Candidate atomsets will qualify as frequent atomsets if they have support greater than user defined minimum support. In [Knobbe et.al, 99] instead of *ExKey*, notion of *foreign key* is used to prune the search space in multi-relational mining. Using a structure called *selection graph*, multi-relational patterns are captured and are evaluated using the WARMR algorithm.

2.6.5 Knowledge Based Algorithms

These algorithms find association rules based on domain knowledge. Little work is done on this category. Algorithm - **Cumulate** [Srikant et.al, 95] - belongs to this category.

2.6.5.1 An Abstract Version of Knowledge Based Algorithms

Figure 2.11 shows abstraction of knowledge based algorithms.

Figure 2.11 Abstraction of knowledge based algorithms

In algorithms - Cumulate - 'is_a' hierarchy is used as domain knowledge. In these algorithms, the original database, D is generalized to D_g and the basic algorithms like Apriori run on D_g to generate generalized association rules.

2.6.6 SQL Based Algorithms

These are algorithms where mining of association rules is done using SQL constructs. Set oriented mining, **SETM** introduced in [Houtsma et.al, 95] is an example. This algorithm basically uses sort-merge strategy. This algorithm shows how general query languages like SQL can be used to generate large itemsets. Similar to Apriori, in each iteration, it generates large itemsets of size k ($k \geq 2$). The iteration process involves sorting the items in table R_{k-1} generated at step k-1, generating candidate relation $\acute{R}_k$ using merge-scan, generating counts from $\acute{R}_k$ and generating table R_k having only large items.

A comparative study of various forms of SQL queries for mining association rules is reported in [Sarawagi et.al, 98]. Various options in SQL-92, SQL-OR, stored procedures

and Cache-mining approach are compared based on cost analysis. Their relative performance depends on number of items, total number of transactions, average length of transaction, etc.

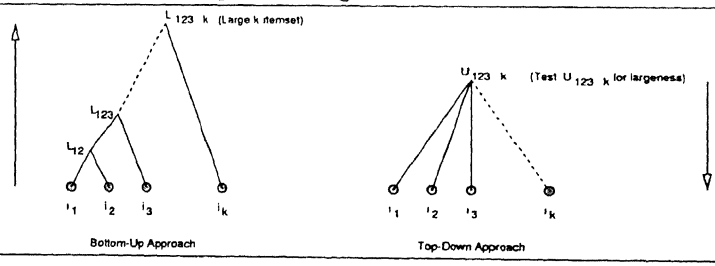
It was pointed out that SQL based mining algorithms are inferior to the ad-hoc file processing algorithms such as Apriori and its variants in terms of speed of mining. However a general idea of "query flock" is introduced in [Tsur et.al, 98]. It is - generate and test model for data mining problems. A flock consists of a parameterized query and a filter that selects certain assignments of values for the parameters by applying a condition to the result of the query for that value assignment. A generalized incremental mining approach including constraint handling using SQL is discussed in [Thomas et.al, 00].

2.6.7 Bottom-up and Top-down Algorithms

In some ARM Algorithms, grouping of itemsets, $i_1, i_2, \ldots, i_k$ to construct large itemset L is done if each of i_j, $(j : 1, 2, \ldots, k)$ has enough support. Such algorithms are called 'bottom-up algorithms'. Level-wise (iteration-based) algorithms like **Apriori** [Agrawal et.al, 93B], **AprioriTid** [Agrawal et.al, 94], **DIC** [Brin et.al, 97] are examples of this category.

On the other hand, if the grouping of itemsets, $i_1, i_2, \ldots, i_k$ to U is done first and later it is checked whether U has enough support or not, then such algorithms are called 'top-down algorithms'. **AprioriSome**, **DynamicSome** [Agrawal et.al, 95A] use this approach to find the association rules. Figure 2.12 shows both the approaches.

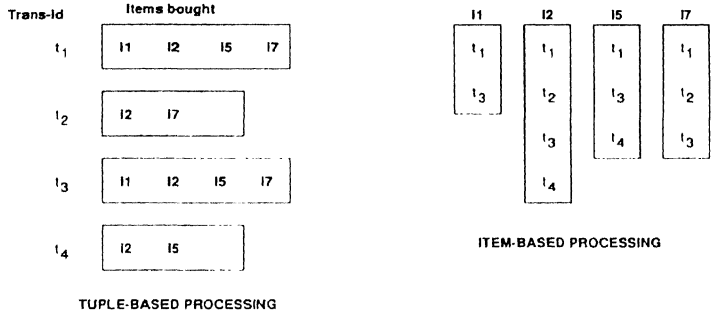
Figure 2.12 Bottom-up Vs. Top-down algorithms



2.6.8 Tuple Based and Item Based Algorithms

Algorithms like **Apriori** [Agrawal et.al, 93B, **FP-tree based algorithm** [Han et.al, 00] use the horizontal data layout where the processing of the transaction database is with respect to the items bought in each transaction. They are called 'tuple-based algorithms'. On the other hand, algorithms like **Partition-based** [Savasere et.al, 95], **Apriori-Tid** [Agrawal et.al, 94], **(Max)Eclat**, **(Max)Clique** [Zaki et.al, 97B], **Viper** [Shenoy et.al, 00] use vertical data layout where the processing of the transaction database is with respect to the transaction ids corresponding to the items. Such algorithms are called 'item-based algorithms'. Figure 2.13 shows both the schemes.

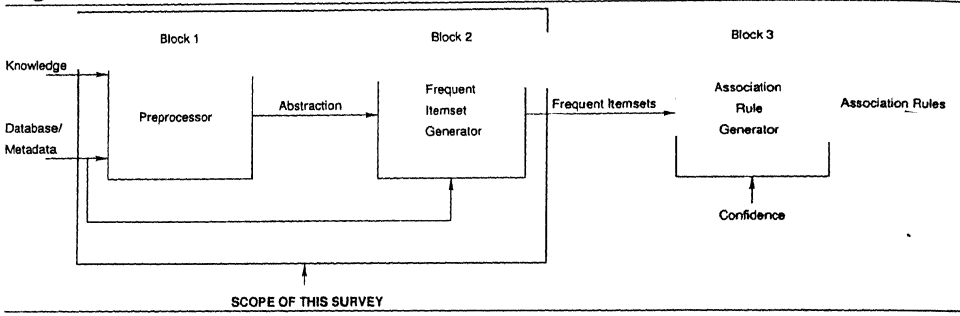
Figure 2.13 Tuple-based (Horizontal) Vs. Item-based (Vertical) algorithms



2.7 Structure of Algorithms for Mining Frequent Itemsets

As described earlier, the problem of mining association rules can be decomposed into two subproblems. The first problem is to find all frequent itemsets and second is to generate association rules using the frequent itemsets found. The process of mining association rules can be described with the help of the block diagram shown in Figure 2.14. Once the frequent itemsets are obtained, generating association rules is reasonably straightforward. In fact it is possible to perform this step online [Aggrawal et.al, 97]. So, we examine only the first subproblem in detail in this review.

Figure 2.14 Process of mining association rules



Frequent itemsets are generated using either the **original database** or its **abstraction** obtained using a **preprocessor**. We describe the preprocessing step, in an abstract manner, based on several variants employed in literature.

2.7.1 Preprocessing

The preprocessing step requires as inputs, **knowledge** and **database/metadata**. This step generates as its output an **abstraction** based on **at most** one scan of the database. This is because the output (any abstraction) of the PREPROCESSOR is generated by using either the entire database once or the metadata. The motivation behind this characterization is that **number of database scans** is a pragmatically important and frequently used parameter in literature on mining association rules. Also, *every algorithm* for generating frequent itemsets can be abstracted using this I/O behaviour.

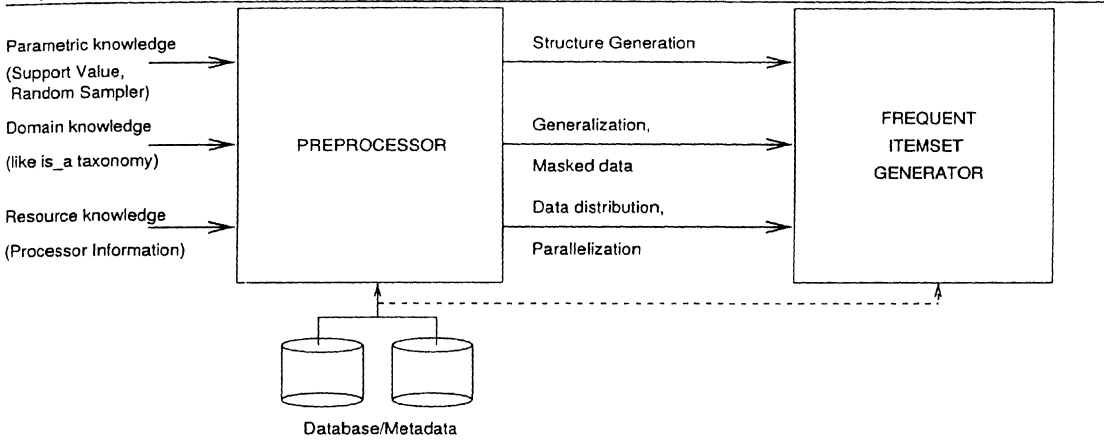
The process of generating frequent itemsets is depicted in Figure 2.15. We explain the preprocessing step in detail below:

Inputs to the preprocessor :

- knowledge
- database/metadata

Knowledge : The different kinds of knowledge that are input to the preprocessor are :

Figure 2.15 Preprocessing step



1. Parametric knowledge : This kind of knowledge is based on the distribution of data. Specific forms of usage of knowledge here are : (i) minimum support value and (ii) random sampling [Toivonen, 96]. Typically the user inputs the minimum support value. Random sampling exploits the distribution of the data to be mined.

2. Domain knowledge : This knowledge is based on the application for which the database is created. One form of knowledge used for mining is *is-a* hierarchy, which connects lower level concepts to higher level concepts.

3. Resource knowledge : This component deals with resources like : number of processors and their distribution, speed of processors, memory available, data distribution across the processors and connectivity among processors.

Database/metadata : Usually a single database [Agrawal et.al, 93B; Han et.al, 92; Han et.al, 93] is input to the preprocessor. This input contains the data used to mine frequent itemsets. In incremental mining like FUP_2 [Devid et.al, 97], the input is the $D\acute{B} \cup db$, where $D\acute{B}$ is the modified version of the original database DB and db is a collection of transactions newly added. $D\acute{B}$ results from DB based on deletion of some items and modification to some transactions.

Output from the preprocessor :

1. Support value along with the database is used for generating **structures** like large 1-itemsets [Agrawal et.al, 93B], locally-large itemsets [Savasere et.al, 95], and lattice structures [Hidber et.al, 99; Zaki et.al, 98]. Random sampler [Toivonen, 96] along with the database is used for generating a partial set of large itemsets which is a **structure**.

2. Domain knowledge like - is_a hierarchy along with the database is used for generating generalized [Srikant et.al, 95]/specialized [Han et.al, 95] itemsets.

3. Resource knowledge along with database(s) in the preprocessing step is used for distribution of data across processors [Eui-Hong et.al, 97; Shintani et.al, 98; Cheung et.al, 96B].

2.7.2 Frequent Itemset Generator

Output of the preprocessor is input to the **frequent itemset generator**. Depending on the **types of abstractions** generated by the preprocessor, algorithms for generating frequent itemsets can be categorized based on **number of scans of database** as follows:

One database scan : Some of the algorithms need to scan the database to generate frequent itemsets based on the abstractions generated by the preprocessor. For example, the algorithm reported in [Han et.al, 00] needs one database scan to construct the FP-tree from large 1-itemsets. Similarly, partition based algorithm [Savasere et.al, 95] needs one scan of the database to generate globally-large itemsets from the locally-large itemsets.

More than one database scan : Algorithms like Apriori [Agrawal et.al, 93B] make use of $(p - 1)$ scans of the database to construct large p-itemsets, from large 1-itemsets, where $p > 2$. Such an algorithm requires more than one database scan to generate frequent itemsets.

2.7.3 Comparative Study of Algorithms with respect to Number of Database Scans

Table 2.1 shows the comparative study of popular algorithms based on total number of database scans required.

Notations used :

D : Database having tuples $t_1, t_2, \ldots, t_n$

I : a set of items $i_1, i_2, \ldots, i_m$

k : is maximal cardinality of a large itemset in D, k $\leq m$

p : number of partitions

r : number of relations in the database

Observations : It is clear from Table 2.1 that majority of the algorithms need 'k' (k > 2) number of database scans; few of them need '2' database scans. It is also observed that

Candidate non-generation algorithms (viz., abstraction based algorithms) use less number of database scans. So, they are *machine-friendly.*

Even though iterative algorithms are *human-friendly* (viz., same steps are repeated in each iteration of the algorithm), they use many number of database scans.

2.8 Summary

We classified association rules based on their structural and semantic properties. We abstracted the behaviour of frequent itemset generating algorithms. Such an abstraction helps in categorizing various association rule mining algorithms. A general frame-work structure of any ARM algorithm is discussed.

Table 2.1 A comparison table : Number of database scans

Algorithm Name	Output from Preprocessor	No. of Database Scans		
		Pre-processor	Frequent Itemset Generator	Total
Apriori[Agrawal et.al,93B]	large 1-itemsets	1	k-1	k
AprioriTid[Agrawal et.al,94]	large 1-itemsets	1	k-1	k
DHP[Park et.al,95]	large 1-itemsets	1	k-1	k
DIC[Brin et.al,97]	large 1-itemsets	1	<(k-1)	< k
AprioriAll/AprioriSome DynamicSome[Agrawal et.al,95]	large 1-itemsets	1	k-1	k
SETM[Houtsma et.al, 95]	large 1-sequences	1	k-1	k
FITI[Tung et.al, 99]	large 1-itemsets	1	k-1	k
FUP[Cheung et.al, 96A]	large 1-itemset in $DB \cup db$ and db	1	k-1	k
FUP$_2$[Cheung et.al, 97]	large 1-itemset in $DB \cup db$ and db	1	k-1	k
Quantitative rules[Srikant et.al, 96]	Mapping of quantitative and category-based values to set of integers	0	k	k
Dense Miner[Roberto et.al,99 A]	Generating initial group for set-enumeration tree	0	k	k
MINWAL[Cai et.al, 98]	large 1-itemsets	1	k-1	k
ML-T1LA[Han et.al, 95]	large 1-itemsets (l[1,1]) for each level l of taxonomy	1	k-1	k
Negative rules[Savasere et.al,98]	large 1-generalized itemsets	1	k-1	k
(Max)Eclat[Zaki et.al,99B]	Prefix-based classes	1	k-1	k
(Max)Clique[Zaki et.al,99B]	Clique-based classes	1	k-1	k
Partition based[Savasere et.al,95]	locally large itemsets	1	1	2
SPINC[Mueller,95]	locally large itemsets	1	2(p-1)/p- 1	2(p-1)/p
AS-CPA[Jun-Lin et.al,98]	locally large itemsets	1	2(p-1)/p- 1	2(p-1)/p
Sampling[Toivonen,96]	partial set of large itemsets	1	1	2
SSAS-CPA[Jun-Lin et.al,98]	partial set of large itemsets	1	2(p-1)/p- 1	2(p-1)/p
FP-tree based[Han et.al,00]	large 1-itemsets	1	1	2
CARMA[Hidber,99]	lattice	1	1	2
Multi-relation based(WARMR)[Luc Dehaspe et.al,97]	Large 1-atomsets	r	k × r	r (k+1)
Cumulate[Srikant et.al,95]	large 1-generalized itemsets	1	k-1	k

Chapter 3

Problem Formulation

3.1 Introduction

In this chapter, we give notations and definitions used, general foundation for association rules and definition of the problem considered in the thesis. We also present the solution methodology proposed and used by us.

Section 3.2 gives definitions and notations used in this thesis. We discuss the general foundation for association rules in section 3.3. This formalism helps in characterizing the generation of association rules from a collection of (one or more) databases. Section 3.4 gives a frame-work for association generation explored in this thesis. Summary of the chapter is provided in section 3.5.

3.2 Notations and Definitions

3.2.1 Databases

Notations :

1. $\mathbb{D}$ is a set of databases written as $\mathbb{D} = \{D_1, D_2, \ldots, D_n\}$, where D_i is the i^{th} database. Each database is either relational, hierarchical or network database. However, for the sake of simplicity, we assume that the hierarchical and network

schemas are converted into relational form.

2. R_j^i is j^{th} relation in i^{th} database D_i, where j = 1,2, ..., r_i.

3. The relation R_j^i, $i = 1, \ldots, n$ and $j = 1, \ldots, r_i$ is defined on attributes $K_{j1}^i, K_{j2}^i, \ldots, K_{jk}^i, A_{j1}^i, A_{j2}^i, \ldots, A_{ja}^i, MVA_{j1}^i, MVA_{j2}^i, \ldots MVA_{jm}$, where the composition of attributes $K_{j1}^i \ldots K_{jk}^i$ represents a *key*, attributes $A_{j1}^i, \ldots A_{ja}^i$ are *single-valued attributes* and attributes $MVA_{j1}^i, \ldots MVA_{jm}^i$ are *multi-valued attributes.*

4. There are three types of attributes which are of interest to our study. They are general attributes (single-valued and multi-valued attributes), common set of attributes and keys.

Definitions :

3.2.1.1 Database schema : It gives a description of the database, which is specified during database design and is not expected to change frequently. For example, schema for a student information system include STUDENT(Name, ID-Number, Address, Dept-Name); GRADE(ID-Number, Course-Number, Grade), etc. **Data dictionary** depicts the schemas.

3.2.1.2 Global and Local schemas: Whenever a complex database - with numerous expected users - is to be designed, individual user groups can independently design their own schemas representing their particular data requirements. This serves to reduce the complexity of the design and to allow each user group to define their preliminary data requirements without undue external influence. These individual schemas are called local schemas. They are then integrated into a global conceptual schema (global schema) that describes the overall data requirements for the database.

3.2.1.3 Domain of an Attribute : Each attribute of a relation is associated with a **domain**, which specifies the set of legal values that can be assigned to the attribute.

For example, in the relation **Emp**, we can specify the domain of Age attribute (i.e., **Domain**(Age)) to be the set of integers between 16 and 70.

3.2.1.4 A Key is a collection of attributes, which is used to link associations *within a* database.

3.2.1.5 A Common set of Attributes (CoAs) is a collection of attributes which belong to *at least* two different databases and the corresponding attributes have the *same* domain in these databases. This type of attributes is used to navigate *across* databases. i.e., to link two or more databases.

Example : The set of attributes (*emp_name*, *emp_address*) in the Employee database and the set of attributes (*cust_name*, *cust_address*) in Customer database are common set of attributes since both have the same domain.

3.2.1.6 Characteristic attributes (CAs) : These are general attributes which participate in finding associations. For a relation R_j^i, a characteristic attribute is denoted as CA_{jx}^i, where x can assume a value in the range 1 to $|R_j^i|$, where $|R_j^i|$ is the number of attributes in R_j^i.

3.2.1.7 Generalization : Generalization is a mapping which is achieved with a help of a tree called generalization tree. The most general value is defined by reserved word 'any' corresponding to the root of the tree [Han et.al, 92]. Each leaf node in the tree is associated with the most specific data value. A recursive generalization process can potentially take us from the most specific data values to 'any'. Note that each such tree is associated with an attribute occurring in a relation of a database. In data mining literature, it is assumed that these generalization trees are provided by knowledge engineers or domain experts. Figure 3.1 shows generalization trees for attributes Age, Address and Salary.

Figure 3.1 Generalization trees

3.2.2 Semantic Network

1. S is the semantic network [Findler, 79], (N, E), where N is the set of nodes and each node corresponds to a 'concept' and E is the set of edges connecting any two nodes.

2. We consider a semantic network to be a Directed Acyclic Graph (DAG). We make this assumption to simplify the structure without any loss of generality.

3. There are three types of concepts. They are Independent concept (IC), Characteristic concepts (CCs), and Dependent concepts (DCs).

4. Each characteristic concept maps to an attribute of a relation in a database.

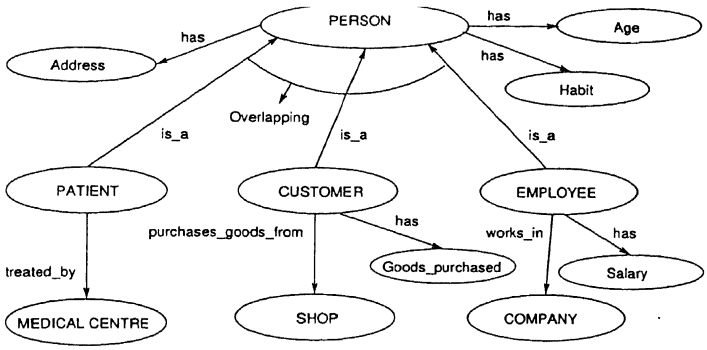
3.2.2.1 Concept : It can be described using a logical expression.

For example, 'salary' is a concept made up of one term and usually is the characteristic of some other concept like 'employee'. 'Employee' on the other hand is a finite conjunction of terms like 'name = RAMA','designation = MANAGER','salary = Rs.22,000', ... ,etc. So, a concept represents an **attribute** or a **relation**.

Example : Semantic network for Person information system:

Figure 3.2 depicts a semantic network showing knowledge of commonly understood personal information. A person works in a company, purchases from a super market and gets treatment from a hospital. Based on the mode of activity, he can be an employee, a customer or a patient. So, the node labeled 'PERSON' represents the generic concept of a person. It can be thought of as possessing properties common to all persons, viz., employee, customer and patient. We call such a concept, **independent concept (dominant entity)**. Since the semantic network which we

Figure 3.2 Semantic network



considered is a DAG, the root of the semantic network is the independent concept (IC). The concepts corresponding to nodes which are connected by 'has' link to other nodes are called **characteristic concepts** (CCs). For example, in Figure 3.2, concept 'Habit' is a characteristic concept. Since CCs are candidates for associations mining activity, they have mapping with **characteristic attributes**. For example, while finding the associations between the values of salary and goods_list, for the characteristic concept salary and goods_purchased in the semantic network, the associated characteristic attributes are *salary* and *goods_list*. Concepts, other than the independent concept which are not characteristic concepts are called **dependent concepts** (DCs). In Figure 3.2, nodes labeled 'PATIENT', 'CUSTOMER' and 'EMPLOYEE' correspond to dependent concepts.

In association mining, we want to find the associations between values of characteristic attributes which correspond to characteristic concepts. CCs may belong to a independent concept, or one or more dependent concepts. If CCs belong to more than one dependent concept, then linking of characteristic concepts is done using the independent concept.

Dependent and independent concepts are described in *one or more* databases. For example, we have the information of a person as an employee in one database that describes his professional information like salary, department where he is working,

etc. We have the information about a person as a customer in another database that may describe his purchase habits. Patient database gives the disease descriptions like disease type, date of admission, etc, of the person as a patient. As a part of personal information, all of these databases have some common information like habit, address, age, sex, etc.

3.3 General Foundation for Association Rules

In this section, we formulate the notion of item, itemset and valuetable which are helpful in characterizing association rules.

The world of reference is a finite collection of databases $\mathbb{D} = \{D_1, D_2, \ldots, D_n\}$. Each database, D_i, $i = 1, \ldots, n$, is a collection of relations $R_1^i, R_2^i, \ldots, R_{r_i}^i$. Each relation, R_j^i, $i = 1, \ldots, n$ and $j = 1, \ldots, r_i$ is defined on attributes, $K_{j1}^i, K_{j2}^i, \ldots, K_{jk}^i, A_{j1}^i, A_{j2}^i, \ldots, A_{ja}^i, MVA_{j1}^i, MVA_{j2}^i, \ldots MVA_{jm}^i$, as described in section 3.2.1.

Definition 3.3.1 Well Formed Formula (wff) : It is made up of one or more atoms connected by logical operators $\wedge$, $\vee$, $\neg$, where atom has one of the following forms [Elmasri et.al, 00] :

1. $R(t)$.
2. $t_i[A_p]$ **op** $t_j[A_q]$.
3. $t_i[A_p]$ **op** c.

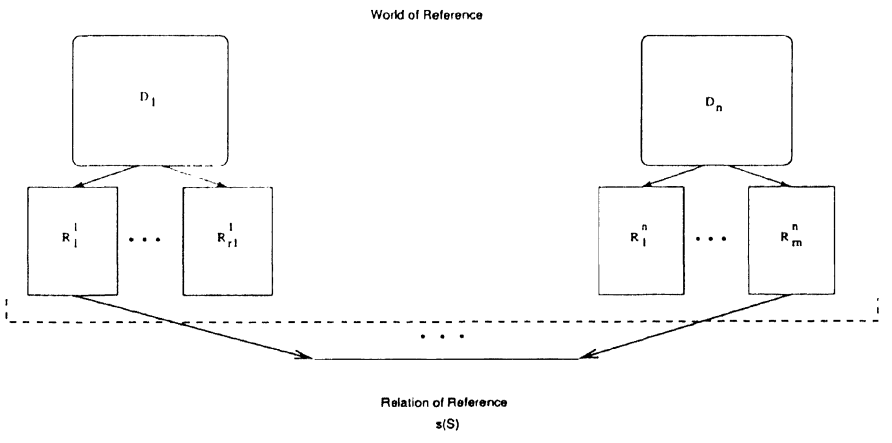
Where **op** is one of the comparison operators from the set $\{=, \neq, <, \geq, <, \leq\}$, c is a constant, R is a relation name, and t, t_i and t_j are tuple variables which range over relation R which has attributes A_p and A_q.

If f_1, f_2 are wffs, then $f_1 \wedge f_2$, $f_1 \vee f_2$ and $\neg f_1$ are also wffs. In addition, two special symbols called **quantifiers** can appear in a wff; these are the universal quantifier ($\forall$) and the existential quantifier ($\exists$).

1. If F is a wff, then so is $(\exists\ t)(F)$. The formula $(\exists\ t)(F)$ is TRUE if the formula F evaluates to TRUE for *some* (at least one) tuple assigned to free occurrences of t in F; otherwise $(\exists\ t)\ (F)$ is FALSE.
2. If F is a wff, then so is $(\forall\ t)(F)$. The formula $(\forall\ t)(F)$ is TRUE if the formula F evaluates to TRUE for *every* tuple (in the universe) assigned to free occurrences of t in F; otherwise $(\forall\ t)\ (F)$ is FALSE.

Definition 3.3.2 Relation of Reference : It is a generated reference relation for mining, and is explicitly specified as $s(S) = \bowtie \mathfrak{r}$. Here, $\mathfrak{r}$ is a set of relevant relations selected from $\mathbb{D}$. **Valuetable** $(\{t \mid \psi(t)\})$ is obtained by applying wff (i.e., $\psi(t)$) over $s(S)$. It is represented as $\kappa = \{t_1^l, t_2^l, \ldots, t_n^l\}$, which is a bag because t_i^l may be equal to t_j^l for some $i, j \in (1 \ldots n)$ and $i \neq j$. Each t_i^l is an ***l*-tuple** and the valuetable is called ***l*-Valuetable**. Each t_i^l is of the form, $< v_{oj_1}^x, v_{pm_2}^y, \ldots, v_{qn_l}^z >$, where the c^{th} entry in t_i^l, $v_{abc}^d \in \textbf{Domain}(A_{ab}^d)$. So, the l-tuple can be viewed as an instantiation of the Cartesian product of l attributes A_{oj}^x, A_{pm}^y, ..., A_{qn}^z. Relation of reference and valuetable are depicted pictorially in Figure 3.3.

Figure 3.3 Setting the mining space



Definition 3.3.3 Sub-tuple : Let t_i^l be an l-tuple of the form $< v_1, v_2, \ldots, v_l >$ which is an instantiation of the Cartesian product of attributes $B_1, B_2, \ldots, B_l$. A k-tuple $< \acute{v}_1, \acute{v}_2, \ldots \acute{v}_k >$ which is an instantiation of the Cartesian product of attributes $A_1, A_2, \ldots, A_k$ is a *sub-tuple* of t_i^l if

1. $\forall i$, $1 \leq i \leq k-1$, if $A_i = B_j$ and $A_{i+1} = B_p$ then, $1 \leq j \leq l$ and $j < p \leq l$. and
2. Projection of $s(S)$ over attributes $A_1, A_2, \ldots, A_k$ has $< \acute{v}_1, \acute{v}_2, \ldots \acute{v}_k >$ as its ^{ith} tuple.

Note that every sub-tuple of t_i^l, based on attributes $A_1, A_2, \ldots A_k$ is the i^{th} tuple in the projection of $s(S)$ over attributes $A_1, A_2, \ldots, A_k$.

The relational database design method is based on the lossless decomposition of relations into normal forms. But such a decomposition hides dependencies that might be useful for mining association rules. The ideal context for mining is the *universal relation.* So, the set of relations built using traditional database design has to be put together into a single relation (i.e., *denormalization* [Elmasri et.al, 00; Addriaans, 99] so that interesting dependencies can be preserved. This is the reason behind our construction of universal relation for mining, called Relation of reference having all required attributes (including multi-valued attributes), whose associations could be interesting.

Relation of reference, $s(S)$, characterizes the value space on which the mining operation is performed. Note that Relation of reference fixes the scope of mining and the wff $\psi(t)$ captures the constraints [Srikant et.al, 97; Roberto et.al, 99A] imposed on the domains of values. In case of un-constrained mining, $\psi(t)$ is logically valid [Mendelson, 64] or a tautology in the propositional sense. In this scenario, the relation of reference, $s(S)$, is the valuetable.

Let κ^k be the set $\{u^k \mid (\exists t)(t \in s) \wedge (u_1 = t[A_i]) \wedge (u_2 = t[A_j]) \wedge \ldots \wedge (u_k = t[A_m]) \wedge \psi(t)\}$. Here each $t_i^k \in \kappa^k$ is a k-tuple and $\psi(t)$ is any wff. Note that by changing $\psi(t)$, we get different k-valuetables.

Consider the Relation of reference, $s(S)$ shown in Table 3.1 constructed from EMPLOYEE and PERSON-HABIT relations. Here, each tuple, t_i^7 is a 7-tuple.

Table 3.1 Relation of reference, $s(S)$

	ssn	name	city	age	designation	salary	habit
t_1	CS001	rajan	bangalore	senior	engineer	high	coffee_drinking
t_2	CS002	john	belgaum	senior	manager	high	tea_drinking
t_3	CS003	baig	bangalore	senior	manager	high	coffee_drinking
t_4	CS004	ramesh	mysore	senior	engineer	low	coffee_drinking
t_5	CS005	rama	mysore	junior	engineer	high	coffee_drinking
t_6	CS006	gopal	mysore	junior	manager	high	tea_drinking

Let $\kappa^2 = \{u^2 \mid (t \in s) \wedge (u_1 = t[\textbf{city}]) \wedge (u_2 = t[\textbf{habit}])\}$. Here, we get $\kappa^2 =$ {<bangalore, coffee_drinking>, <belgaum, tea_drinking>, <bangalore, coffee_drinking>, <mysore, coffee_drinking> <mysore, coffee_drinking> <mysore, tea_drinking>}; note that each tuple, t_i^2 of κ^2 is a 2-tuple. Here, κ^2 is a valuetable and the tuple <bangalore, coffee_drinking> is a sub-tuple of t_1^7 and t_3^7.

Definition 3.3.4 Item : An item is a value of an attribute under consideration. For example, 'coke' is a value of the attribute 'Soft_drink'. So we treat *coke* as an item. Note that here, 'item' is used in a more general sense. In transaction database based ARM activity, 'item' is a value of the attribute 'items_purchased'.

Definition 3.3.5 p-itemset : It is a set of distinct p items. For example, {bread, milk, jam} is a 3-itemset. Similarly, {bangalore, coffee_drinking} is a 2-itemset.

There are situations where generalized items or hierarchy of items [Elmasri et.al, 00] are encountered. These are defined using domain knowledge. Generalized items are subsets of the domain of an attribute. For example, it is possible to partition the domain of an attribute into subsets so that each subset is associated with a label. For example, the domain {21...65} of attribute **Age** can be partitioned into two subsets that are labelled **junior** and **senior** as follows :

junior – {21...40} (age in the range 21 to 40)

senior – {41...65} (age in the range 41 to 65)

It is possible to view these labels as **generalized** values. Generalized items are abstractions of generalized values. We call both items and generalized items as items to be consistent with their use in the literature.

Every association rule must satisfy two user specified constraints, one is *minimum support* and the other is *minimum confidence*. The support of a rule $\mathfrak{A} \Longrightarrow \mathfrak{C}$ is defined as the fraction of tuples that contain both $\mathfrak{A}$ and $\mathfrak{C}$, where $\mathfrak{A}$ and $\mathfrak{C}$ are sub-tuples of tuples in $s(S)$ satisfying wffs based on $s(S)$. In otherwords, support is obtained by counting the number of tuples of which both $\mathfrak{A}$ and $\mathfrak{C}$ are sub-tuples, which we represent $(\mathfrak{A}+\mathfrak{C})$ under condition $\psi(t)$ and is divided by total number of tuples in $s(S)$.

Given an j-tuple, $\mathfrak{X} = \{x_1, x_2, \ldots, x_j\}$, where, each $x_i \in$ **Domain**(A_{pi}) $(1 \leq i \leq j)$, we define **Count**$(\mathfrak{X})$ as :

$\{\text{Count}(t) \mid (t \in s(S)) \wedge (\psi(t)) \wedge (t[A_{p1}] = x_1) \wedge (t[A_{p2}] = x_2) \wedge \ldots \wedge (t[A_{pj}] = x_j)\}$

We explain the idea of **Count** with the following example. Consider the Relation of reference, $s(S)$ shown in Table 3.1. The output of **Count**(mysore) on $s(S)$, where $\psi(t)$ = $\{t[(\textbf{age}] = \text{junior}) \wedge (t[\textbf{salary}] \neq \text{low})\}$ is 2. Note that here, A_{p1} is **city** and x_1 is 'mysore'.

Definition 3.3.6 Frequent (large) tuple : A sub-tuple, t of one or more tuples in $s(S)$ is said to be frequent (large), if

$$\frac{\textbf{Count(t)}}{|s(S)|} \geq \sigma$$

where, $|s(S)|$ is the number of tuples in Relation of reference $s(S)$, and σ is the user defined minimum support.

For example, consider the relation of reference, $s(S)$ shown in Table 3.1. If σ = 0.3, the sub-tuple <bangalore, coffee_drinking> is a frequent tuple because, **Count**(bangalore, coffee_drinking)/$|s(S)|$ = 2/6 > 0.3.

Definition 3.3.7 Candidate tuple : Some ARM algorithms generate the possibly frequent tuples of length $(k+1)$ iteratively, without looking at the data, from the set of frequent tuples of length k (for $k > 1$). Such iteratively generated tuples are called *candidate tuples.*

We use 'itemset' and 'tuple' interchangeably, if the context permits such a usage. For example, if the 'relation of reference' considered is the *transaction database*, then the 'valuetable' is the projection of transaction database on attribute *items_purchased*; and a 'tuple' is an *itemset.* Similarly, if the valuetable has only values corresponding to *a* multi-valued attribute, then a 'tuple' represents a 'set of values', where each value belongs to the domain of the multi-valued attribute.

Confidence is defined as the ratio, $\mathbf{Count}(\mathfrak{A} + \mathfrak{C})/\mathbf{Count}(\mathfrak{A})$. Here, '+' indicates the union of tuples, $\mathfrak{A}$ and $\mathfrak{C}$. The goal is to find all rules that satisfy the user defined minimum support (σ) and minimum confidence (c).

Mining association rules consists of two parts.

1. Discovery of all *frequent tuples*, from either $s(S)$ or its abstraction using user defined minimum support value (σ).

2. Use the frequent tuples to generate association rules using user defined minimum confidence factor (c), i.e., constructing rules of the form $\mathfrak{A} \Longrightarrow \mathfrak{C}$ from the frequent tuples, such that ratio $\mathbf{Count}(\mathfrak{A} + \mathfrak{C})/\mathbf{Count}(\mathfrak{A})$ is greater than or equal to c.

3.3.8 Association : It is the mapping between tuples. In general, they may be viewed as *n-ary* associations (where, $n \geq 2$) between tuples which are the instantiation of the Cartesian product of attributes belonging to different relations/databases. More specifically an n-ary association can be classified as :

1. a *binary* association (where, $n = 2$): which gives a relationship between the values of *two* characteristic attributes, CA^k_{ix} and CA^l_{jy}. If $(i = j, x = y$ and $k = l)$, then the association rule is between two *distinct* set of values corresponding to *the same* characteristic attribute, CA^k_{ix}. For example, in transaction database

mining [Agrawal et.al, 93B], we may generate an association rule of the form $v_i \Longrightarrow v_j$, where v_i and v_j are set of values corresponding to the attribute, *items_purchased*.

2. a *non-binary* association (where, $n > 2$): which gives a relationship between sets of tuples, which are the instantiation of the Cartesian product of n characteristic attributes, $CA_{i1}^{j1}, CA_{i2}^{j2}, \ldots CA_{in}^{jl}$, where, each of the characteristic attributes may belong to different relations/databases.

Definition 3.3.9 Association rule : It is an *association* between $\mathfrak{A}$ and $\mathfrak{C}$, whose support and confidence are greater than or equal to σ and c respectively. This association is represented by $\mathfrak{A} \Longrightarrow \mathfrak{C}$. Here, $\mathfrak{A}$ and $\mathfrak{C}$ are sub-tuples of tuples in $s(S)$.

NOTE :

1. If the values in LHS and RHS of the association rule belong to a single valued attribute with domain = {0, 1}, (where 0 indicates the *absence* and 1 indicates the *presence* of the value), then there is no need to specify the value along with the attribute name. For example, an association rule, '(**coffee_powder** = 1) $\Longrightarrow$ (**sugar** = 1) $\wedge$ (**milk** = 1)' can be simplified as '**coffee_powder** $\Longrightarrow$ **sugar,milk**'.

2. Let *High_salary* be an attribute with the domain, {25K ... 50K} and let *Heart_disease* be an attribute with the domain, {Congenital-Cardiovascular, High-BP, Heart-attack}. Based on the notion of generalized association rule [Srikant et.al, 95], one can generate the rule, *High_salary* $\Longrightarrow$ *Heart_disease*. This is the association rule where both antecedent and consequent are attributes.

3. An association of the form $\mathfrak{A} \Longrightarrow \mathfrak{C}$ need not be *symmetric*. If we form the rule $\mathfrak{A} \Longrightarrow \mathfrak{C}$, it means that tuples $\mathfrak{A}$, $\mathfrak{C}$ and $\mathfrak{A} + \mathfrak{C}$ are all frequent. Further, the ratio, **Count**$(\mathfrak{A} + \mathfrak{C})$/**Count**$(\mathfrak{A})$ exceeds the minimum confidence c, but the

ratio $\mathbf{Count}(\mathfrak{A}+\mathfrak{C})/\mathbf{Count}(\mathfrak{C})$ may be smaller than the minimum confidence, c. So it is possible that the mining algorithm may generate $\mathfrak{A} \Longrightarrow \mathfrak{C}$ but not $\mathfrak{C} \Longrightarrow \mathfrak{A}$.

Majority of ARM algorithms are designed to mine a single transaction database. In these cases, the relation of reference is the transaction database itself. However, in general, a database and a relation of reference derived from database/s *need not* be the same. This is particularly true when the ARM activity involves many databases and there are constraints in choosing the attributes from these databases.

3.4 Problem Definition

One way of discovering associations involving multiple databases is to exhaustively search different permutations of a collection of relations (candidates) belonging to different databases. This scheme has a time complexity of $\bigcirc([n \times a]^q)$, where q is the number of relations to be considered, n is the average number of tuples in each relation and a is the average number of columns in each relation. So it is computationally prohibitive to pursue this scheme. One can reduce the time complexity significantly, in the above case, by using domain knowledge to identify relevant databases semantically.

In our approach, we investigate a mechanism to discover associations between tuples which are instantiations of the Cartesian product of attributes occurring in one or more relations present in one or more databases. This process is guided by knowledge in the form of a semantic network. The semantic network is used to identify and order the relevant databases. Once the databases are identified, relation of reference and hence valuetable can be generated explicitly or implicitly. In implicit way of generating valuetable, attribute mapping information present in the data dictionary is used to generate the navigation path. Using this navigation path, a structure is constructed which can be viewed as a valuetable and is mined for associations. It is also possible to generate a structure based on explicit valuetable alone or along with the explicit knowledge in the form of semantic network.

In this work, we used several databases and knowledge in the form of a semantic network. With these inputs we give an algorithm to generate associations.

Input :

1. A semantic network $\mathcal{S}$; which is a DAG, used to identify relevant databases and put them in order. Ordering of databases is based on path made up of independent/dependent concepts between the characteristic concepts which correspond to the characteristic attributes. In general, characteristic attributes $\mathfrak{CA}_{\mathfrak{A}}$ and $\mathfrak{CA}_{\mathfrak{B}}$ are of the form $A_1 \times A_2 \times \cdots \times A_{n1}$ and $B_1 \times B_2 \times \cdots \times B_{n2}$ respectively, where $n1$, $n2 \geq 1$. We want to find the associations between tuples which are instantiations of Cartesian product of $\mathfrak{CA}_{\mathfrak{A}}$ and $\mathfrak{CA}_{\mathfrak{B}}$.
2. A collection of n databases $D_1, D_2, \ldots D_n$. Every one of these databases can be described by a collection of nodes in $\mathcal{S}$.

Output :

Associations between tuples which are instantiations of Cartesian product of $\mathfrak{CA}_{\mathfrak{A}}$ and $\mathfrak{CA}_{\mathfrak{B}}$.

Solution steps :

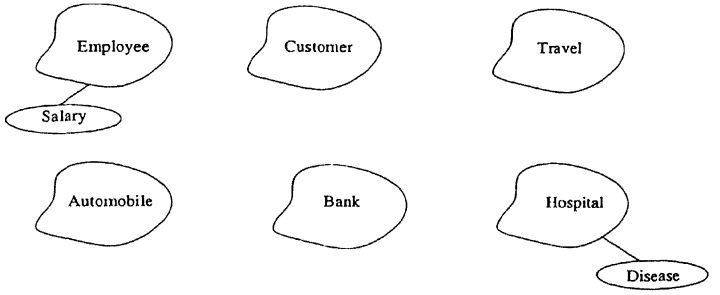
1. *Requirement collection*

   To make best use of data mining, one should identify the goal very clearly. Requirement collection should be done based on a specific goal. For example, to mine in **person information system**, one may identify the requirements as shown in Figure 3.4. Requirement collection is important if the data for mining is not available as a single source. In this thesis, we assume that requirement collection is available as input in the form of a semantic network ($\mathcal{S}$).

2. *Selection of databases*

   This step consists of two tasks :

Figure 3.4 Requirement collection



(a) Identification of concepts : Instead of finding all possible associations, one can specify two sets of characteristic concepts (CC_1 and CC_2) as input. For example, in Figure 3.4, *Salary* and *Disease* are the specified characteristic concepts. Here, there is a need to find associations between the values of corresponding characteristic attributes. In such a case, it is sufficient to find all possible paths based on independent / dependent concepts in ($\mathcal{S}$) which connect CC_1 and CC_2. Otherwise, all independent / dependent concepts specified in $\mathcal{S}$ are treated as 'identified concepts'.

(b) Database identification : For each identified concept find the relevant set of databases using data dictionary.

3. *Schema translation*

This is an optional step. If the databases are heterogeneous, then their 'schemas' may not be in the same data model. In such cases, they should be translated to the relational data model. This process is called schema translation.

4. *Identification of attributes and relations*

We use the idea of *schema integration* to identify the relevant attributes and relations in the selected set of databases. Schema integration methodology is used to integrate local schemas to form the global conceptual schema. In order to obtain the global conceptual schema, there is a need to find which schemas are equivalent, which schemas are disjoint and which schemas are subset of one another. One of the

requirements to do this is to identify the equivalence between the attributes. We use this requirement to identify and order the relations. More specifically, relations are ordered in each database using key/common set of attributes. Primary and foreign keys are used to find the relationship between the relations *in* a database. Common set of attributes are used to find the relationship between relations *across* the databases.

Steps (1) to (4) in that order constitute *data selection* in the KDD process.

5. *Generation of valuetable*

Valuetable is the domain for mining associations. Using the relations/attributes selected, valuetable is generated explicitly or implicitly. Explicit (which is conventional) way of generating valuetable is discussed in section 3.3. Implicit way of generating valuetable is done by constructing structures. Chapters 4, 6 and 7 discuss schemes for generating valuetable implicitly. Structures can also be generated from the explicit valuetable. Chapter 5 and and also the chapter 6 discuss schemes to generate structures based on explicit valuetable. Using the structures, discovery of associations is more efficient because these structures are the abstractions of the database(s) which represent the data in a *compact* form and are *appropriate* for association finding.

Structures are generated in the following way :

- Based on data : Structures discussed in Chapter 4, Chapter 6 and Chapter 7 are constructed based on data.
- Based on explicit knowledge : Structures discussed in Chapter 5 employs explicit knowledge.

6. *Generation of associations*

From the structures, binary or non-binary associations can be generated. Using the associations discovered, based on user specified threshold values (viz., minimum

support value and minimum confidence value) association rules [Agrawal et.al, 93B, Agrawal et.al, 94] can be generated.

For the sake of simplicity consider the associations between values v^x_{aij} and v^y_{bkl}. Depending on the attributes from where v^x_{aij} and v^y_{bkl} are selected, we have following *four* categories of associations.

(a) $(a = b)$ and $(i = k)$ and $(x = y)$: i.e., associations between the values of same attribute which belongs to same relation in the same database. This leads to associations of the form $fever \Longrightarrow headache$ over the attribute *disease*. Conventional association rules belong to this category.

(b) $(a \neq b)$ and $(i = k)$ and $(x = y)$: i.e., associations between the values of different attributes which belong to the same relation. This leads to associations of the form (*grade* = 'senior') $\Longrightarrow$ (*salary* = 'high'), where *grade* and *salary* are two different attributes in the relation EMPLOYEE.

(c) $(a \neq b)$ and $(i \neq k)$ and $(x = y)$: i.e., associations between values of two different attributes which belong to different relations which in turn belong to the same database. An example for this scenario is (*age* = 'range[20-30]') $\Longrightarrow$ (*project-type* = 'programming-oriented'). Here, *age* and *project-type* are two different attributes belonging to two different relations say EMP_PERMANENT_INFORMATION and PROJECT_INFORMATION of a database.

(d) $(a \neq b)$ and $(i \neq k)$ and $(x \neq y)$: i.e., associations between values of two different attributes which belong to relations of different databases. An example for this scenario is (*salary* = 'high') $\Longrightarrow$ (*disease* = 'heart-related disease'). Here, *salary* and *disease* are attributes belonging to different relations, say, EMPLOYEE and PATIENT-DISEASE and also different databases say COMPANY-INFORMATION-SYSTEM and PUBLIC-HEALTH-CENTER.

Two types of associations are discussed in this thesis.

- Based on co-occurrence of values in the same tuple : Chapters 4, 5 and 6 discuss this type of associations.

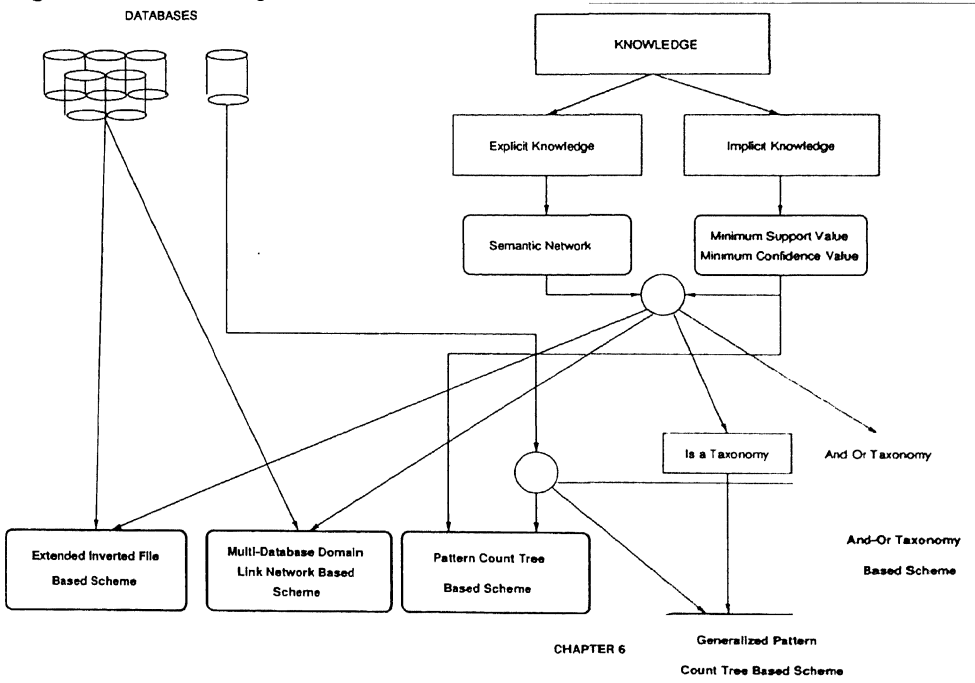
- Not based on co-occurrence of values in the same tuple : Chapter 7 discusses this type of associations.

In this thesis, we propose schemes which generate intermediate structures (abstractions) for mining associations; the associations are in binary or non-binary form.

3.4.1 Flow Diagrams of Proposed Schemes

Flow diagram shown in Figure 3.5 gives schemes which we propose and implement in this thesis for association generation.

Figure 3.5 Flow diagram of schemes



3.5 Summary

In this chapter, we discussed notations and definitions used in this thesis. We formalized the notions of *item*, *valuetable* and *association rule*. We defined *relation of reference*,

which characterizes the value space on which mining is performed. Such a value space is general enough and adequate for any mining activity. We also provided a solution methodology for finding associations across multiple databases.

Chapter 4

Association Generation Using Extended Inverted File

4.1 Introduction

In this chapter, we propose a scheme which uses both explicit and implicit knowledge and a collection of databases as input [cf. 3.4.1] for discovering associations.

In real life, we may have a large collection of data that is stored in several databases in an organized manner. There could be hidden associations among various parts of this data even when it is stored in an organized manner. For example, we have salary information of employees in **Employee** database and an employee when viewed as a customer, we have goods that he purchased in the **Customer** database. There may be associations between salary values and set of goods purchased like *89% of employees having a salary in the range* $(US\$10,000 - US\$15,000)$ *buy Toyota.* This activity of generating association rules between set of values of attributes needs a scheme that helps in navigating across databases. We call the association rules thus generated *inter-database association rules.* In this chapter, we propose a scheme for inter-database association rule mining.

In our approach, we investigate a mechanism to generate association rules between tuples which are instantiations of Cartesian product of attributes under consideration occurring in relations present in different databases. This process is guided by domain

knowledge in the form of a semantic network. The semantic network is used to identify and order the relevant databases. Once the databases are identified, attribute mapping information present in the data dictionary [Elmasri et.al, 00] is used to generate the navigation path. Using this navigation path, a structure is constructed which semantically partitions the resultant relation. In order to discover frequent tuples, we need to scan this resultant relation *only once.* Since each frequent table is a consequent of a potential association rule, there is no more processing required for the generation of association rule. The main contribution of the chapter is :

- *To propose inter-database association rule mining activity and provide a scheme for carrying out this activity.*

  Our contributions reported in this chapter to achieve this major goal are

  - *Generation of data structures to find associations between tuples which semantically partition the resultant database.*
  - *Showing that the proposed approach generates a semantically-partitioned resultant database. Such a partition helps in generating frequent tuples by scanning the resultant database only once.*
  - *It is possible to generate the association rules directly from the frequent tuples; association rule generation is a single step process.*

The remaining part of the chapter is organized as follows. Problem definition is given in section 4.2. Solution methodology is described in section 4.3. Section 4.4 deals with the properties of semantic partitions created by our solution methodology. The theoretical study of disk storage space and disk access time requirements to generate the proposed structure for discovering associations are discussed in section 4.5. The experimental results are discussed in section 4.6. Section 4.7 summarizes our study.

4.2 Problem Definition

In this work, we use several databases and a semantic network.

Input :

1. Two characteristic concepts (CCs) A and B occurring in the given semantic network, $\mathcal{S}$. [For simplicity, we considered two CCs. But it is possible to consider more than two CCs.]
2. A collection, $\mathbf{n}$, of databases $D_1, D_2, \ldots, D_n$. Every one of these databases can be described by a collection of nodes in $\mathcal{S}$.
3. Generalization tree, T_A of an attribute corresponding to the concept A and a level value l. (The level value gives up to what level we have to do the generalization). Let ρ be the number of generalized values at level l.
4. User defined minimum support, σ and minimum confidence c.

Output :

One or more associations between *values* - v^x_{aij} and v^y_{bkl}, where $v^x_{aij} \in \mathbf{Domain}(A^x_{ai})$ and $v^y_{bkl} \in \mathbf{Domain}(B^x_{bk})$. Here, A^x_{ai} is a characteristic attribute corresponding to the characteristic concept A and B^y_{bk} is a characteristic attribute corresponding to the characteristic concept B. So, the problem is to obtain an association rule of the form $v^x_{aij} \Longrightarrow v^y_{bkl}$.

4.3 Solution Methodology

In this section we give a detailed methodology to solve the problem specified in section 4.2.

Algorithm

We employ an algorithm to find the **Output** from the **Inputs** specified in section 4.2. This algorithm which we call Association Mining Algorithm Across Multi-databases ($AMAAM$) is described below.

We give an example that runs across various components of **AMAAM** to illustrate the behaviour of each of the components.

AMAAM :

It consists of following components.

I. Concept Identification (CI) :

INPUT : A,B and the semantic network, $\mathcal{S} = (N,E)$.

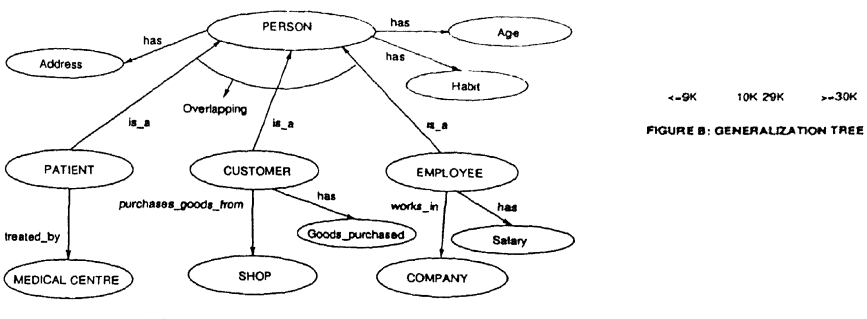
OUTPUT : Concepts $C_1, C_2, \ldots, C_n$ linking A and B in the semantic network.

STEPS :

1. Get a node (concept) C_1, from $\mathcal{S}$, where $(C_1,A) \in E$.
2. Get a node (concept) C_n, from $\mathcal{S}$, where $(C_n,B) \in E$.
3. Output all possible paths in $\mathcal{S}$ each of which connects C_1 to C_n.
4. For each path, execute components **II** to **IV** in $AMAAM$.
5. Merge the frequent tuples generated in the semantic partitions due to each output of **CI** pertaining to a generalized value of a characteristic attribute which maps to the characteristic concept A.

Explanation : Let us consider that we want to find the associations between two characteristic concepts *Salary* and *Goods_purchased*. Let the given semantic network be as shown in Figure 4.1A and let generalization tree for the characteristic concept *Salary* be as shown in Figure 4.1B.

Figure 4.1 Semantic network



Using these inputs, the algorithm **CI** finds independent concepts and dependent concepts which connect *Salary* and *Goods_purchased.* They are EMPLOYEE, PERSON and CUSTOMER.

II. Relation (Schema) Identification and Linking Process (RILP):

INPUT : Concepts $C_1, C_2, \ldots, C_n$

OUTPUT : Relations $\mathfrak{R}_1, \mathfrak{R}_2, \ldots, \mathfrak{R}_m$ which are relevant for linking; and attribute corresponding to A is in $\mathfrak{R}_1$ and that of B is in $\mathfrak{R}_m$.

STEPS :

1. For each C_i, find set, $\mathfrak{D}_i$, of relevant databases using data dictionary.
2. Find a database, call it $D_1 \in \mathfrak{D}_1$, such that one of its relations, call it R_1^i, has attribute A_{1i}^i mapping to the characteristic concept A and call it CA_A.

   Similarly, find a database, call it $D_L \in \mathfrak{D}_n$, such that one of its relations, call it $R_1^{i'}$, has attribute $A_{1j}^{i'}$ mapping to the characteristic concept B and call it CA_B.

   NOTE : If more than one relation has attributes that map to the characteristic concept, one can construct a single relation by properly joining relations using key.
3. Order the databases selected in step 1 as $D_1, D_2, \ldots, D_L$ in the following way:
   - D_1 is the database that has relations containing CA_A and common set of attributes between D_1 and D_2 (i.e., CoA_{12}).
   - D_L is the database that has relations containing CA_B and common set of attributes between D_{L-1} and D_L (i.e., $CoA_{L,L-1}$).
   - Remaining, D_is ($i > 1$ and $i < L$) have relations containing common set of attributes between D_{i-1} and D_i (i.e., $CoA_{i,i-1}$) and common set of attributes between D_i and D_{i+1} (i.e., $CoA_{i,i+1}$).

NOTE : Determination of common set of attributes (CoAs) between the databases is based on attribute equivalence property discussed in schema integration methodology [Ananthanarayana, 95; Larson et.al, 86]. We assume that data dictionary has all CoAs.

4. For each D_i, generate link relations using the algorithm, *Gen_Link_Relations* (D_i,$attr_i$,$attr_j$) as follows :

   (a) For D_1 (i.e., database having attribute pertaining to the antecedent part of association rule), *Gen_Link_Relations*(D_1,CA_A,CoA_{12}) returns R_1^i, R_2^i,...,R_x^i.

   (b) For D_L (i.e., database having attribute pertaining to the consequent part of association rule), *Gen_Link_Relations*(D_L,CA_B,$CoA_{L,L-1}$) returns R_1^L, R_2^L,...,R_y^L.

   (c) For all other D_i ($i > 1$ and $i < L$), *Gen_Link_Relations* (D_i,$CoA_{i,i-1}$,$CoA_{i,i+1}$) returns R_1^i, R_2^i,...,R_z^i.

5. Let $R_1^i, R_2^i, \ldots, R_x^i, R_1^i, R_2^i, \ldots, R_z^i, R_1^i, R_2^i, \ldots, R_y^i$ be m relations generated from step 4. The whole linking process based on keys/CoAs among these relations is shown in figure 4.2.

6. For simplicity, rename these m relations as $\mathfrak{R}_1, \mathfrak{R}_2, \ldots, \mathfrak{R}_m$.

The algorithm for *Gen_Link_Relations*(D_i,$attr_1$,$attr_j$) follows.

Algorithm *Gen_Link_Relations*(D_i,$attr_i$,$attr_j$)

begin

Let $R_1^i, R_2^i, \ldots, R_{ri}^i$ be the set of relations (schemas) in D_i such that

R_1^i has $< attr_i, K_1^i, \ldots >$ attributes.

R_2^i has $< K_1^i, K_2^i, \ldots >$ attributes.

$\vdots$

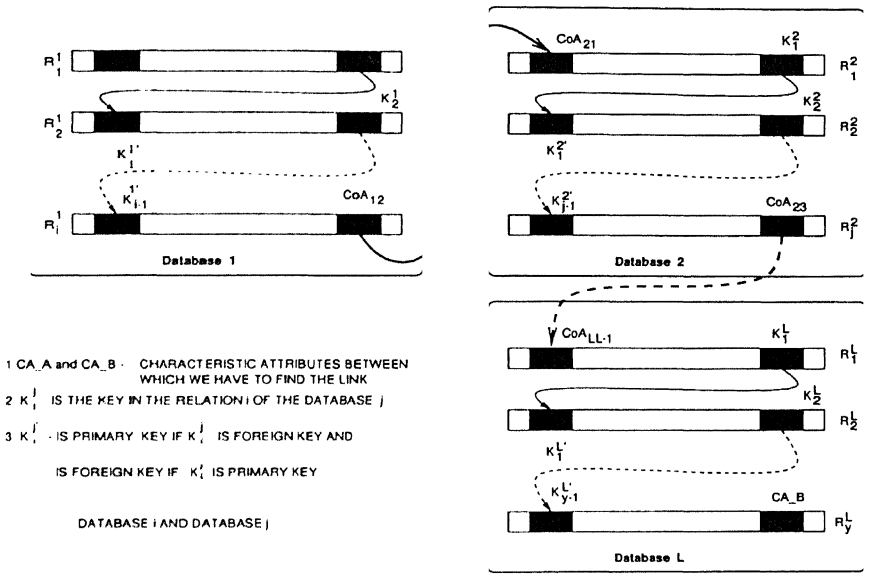
R_{ri}^i has $< K_{ri-1}^i, K_{ri}^i, attr_j \ldots >$ attributes.

Return($R_1^i, R_2^i, \ldots, R_{ri}^i$)

end.

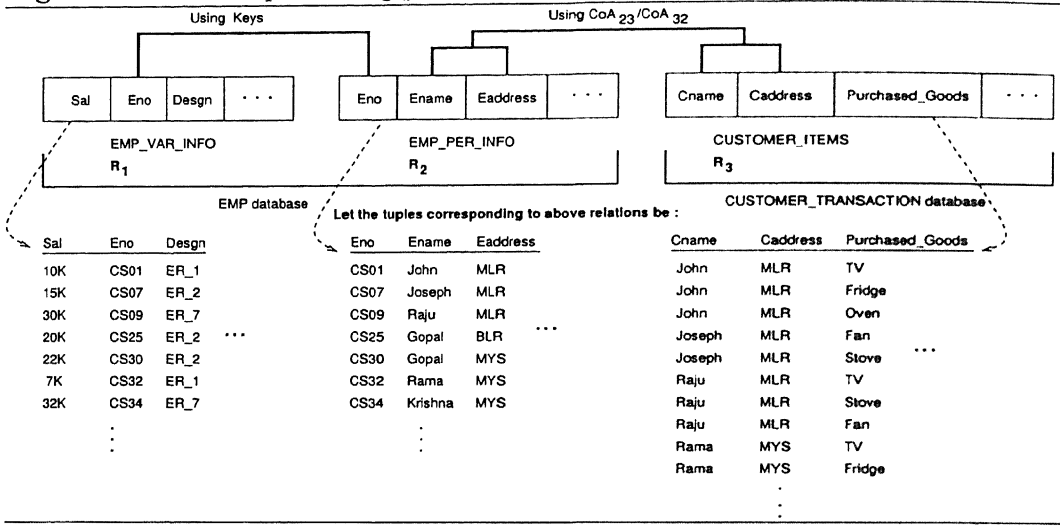
NOTE : K_j^i is key. K_j^i is foreign key if K_j^i is primary key. K_j^i is primary key if K_j^i is foreign key.

Figure 4.2 Linking process



Explanation : The algorithm **RILP** first finds the databases corresponding to the concepts EMPLOYEE, PERSON and CUSTOMER. Let the databases identified be EMP and CUSTOMER_TRANSACTION (the PERSON concept is shared by these two databases). The algorithm next identifies the relation(s) in such a way that first relation in the EMP database should have the attribute corresponding to the characteristic concept *Salary*; let it be *Sal* and the last relation in the CUSTOMER_TRANSACTION database should have the attribute corresponding to the characteristic concept *Goods_purchased*; let it be *Purchased_Goods*. The other relations between the two databases are identified such that they link these two relations. Let the relations identified in that order be as shown in Figure 4.3. The solid lines in Figure 4.3 show the navigation path to connect the attributes *Sal* and *Purchased_Goods*.

Figure 4.3 An example linking process



III. Building an Extended Inverted File (EIF) :

We use an extended version of the well-known "inverted file structure [Trembley et.al, 76]" which we call *extended inverted file (EIF)*. EIF is an extension of inverted file in the sense that it abstracts data present in multiple relations and stores it in a compact form.

Building an EIF consists of following three components.

III-A. Trimming and Preprocessing for Indexing (TPI):

INPUT : $\mathfrak{R}_1, \mathfrak{R}_2, \ldots, \mathfrak{R}_m$.

OUTPUT : $\mathbb{R}_1, \mathbb{R}_2, \ldots, \mathbb{R}_m$ which have the linking attributes; attribute corresponding to A is present in $\mathbb{R}_1$ and that corresponding to B is present in $\mathbb{R}_m$.

STEPS :

1. For each $\mathfrak{R}_i$, construct trimmed relation $\mathbb{R}_i$ as follows :
   - if $(i = 1)$ then

     $\mathbb{R}_i$ has $< CA\_A, C_{i,i+1}, Count >$ attributes.

Where $C_{i,i+1}$ is

- a key if $\mathbb{R}_{i+1}$ is obtained from the same database as $\mathbb{R}_i$, in which case navigation is through a key.
- a common set of attributes if $\mathbb{R}_{i+1}$ is obtained from a different database than that of $\mathbb{R}_i$, in which case navigation is through a common set of attributes.

- if $(i = m)$ then

  $\mathbb{R}_i$ has $< CA\_B, C_{i,i-1}, Count >$ attributes.

  Where $C_{i,i-1}$ is

  - a key if $\mathbb{R}_{i-1}$ is from the same database as $\mathbb{R}_i$, in which case navigation is through a key.
  - a common set of attributes if $\mathbb{R}_{i-1}$ is obtained from a different database than that of $\mathbb{R}_i$, in which case navigation is through a common set of attributes.

- else

  $\mathbb{R}_i$ has $< C_{i,i-1}, C_{i,i+1}, Count >$ attributes.

  Where each $C_{i,i-1}$ and $C_{i,i+1}$ is either a key or a common set of attributes.

NOTE: In the above representation, the attribute *Count* has a value corresponding to the number of times a tuple, based on other attributes (i.e., excluding *Count* field) in relation $\mathbb{R}_i$, occurred.

2. For $\mathbb{R}_1$,

   (a) Generalize the values corresponding to attribute CA_A using generalization tree T_A and generalization level value l.

   (b) Update the count field value of the tuple by adding its value with value in *Count* field of other tuples if they have same values for other attributes (i.e., excluding *Count* field) in relation, $\mathbb{R}_1$. Remove the latter tuples.

   (c) Let $CA\_COUNT(i)$, $1 \leq i \leq \rho$ be the total count for each generalized value of CA_A in $\mathbb{R}_1$. Note that if $CA\_COUNT(i)/|\mathbb{R}_1| < \sigma$ for $1 \leq i \leq \rho$,

then no association rules exist for generalization value i of CA_A. So, in later steps, processing w.r.t generalization value i is not considered.

Explanation : The algorithm **TPI** leads to the construction of three trimmed relations, $\mathbb{R}_1, \mathbb{R}_2$ and $\mathbb{R}_3$ which are shown in Figure 4.4 from EMP_VAR_INFO, EMP_PER_INFO and CUSTOMER_ITEMS shown in Figure 4.3. Note that the *Sal* field value of tuples in $\mathbb{R}_1$ is generalized using the generalization tree shown in Figure 4.1B. Here, $COUNT\_CA(1) = 2$, $COUNT\_CA(2) = 3$ and $COUNT\_CA(3) = 2$.

Figure 4.4 Trimmed relations for the example relations shown in Figure 4.3

IR_1

Sal	Eno	Count
1	CS01	1
2	CS07	1
3	CS09	1
2	CS25	1
2	CS30	1
1	CS32	1
3	CS34	1
...		

IR_2

Eno	Ename	Eaddr	Count
CS01	John	MLR	1
CS07	Joseph	MLR	1
CS09	Raju	MLR	1
CS25	Gopal	BLR	1
CS30	Gopal	MYS	1
CS32	Rama	MYS	1
CS34	Krishna	MYS	1
	...		

IR_3

Cname	Caddr	Purchased_Goods	Count
John	MLR	TV	1
John	MLR	Fridge	1
John	MLR	Oven	1
Joseph	MLR	Fan	1
Joseph	MLR	Stove	1
Raju	MLR	TV	1
Raju	MLR	Stove	1
Raju	MLR	Fan	1
Rama	MYS	TV	1
Rama	MYS	Fridge	1

III-B. Extended Inverted File Generation (EIFG):

INPUT : $\mathbb{R}_1, \mathbb{R}_2, \ldots, \mathbb{l}$

OUTPUT : Extended Inverted File

Data structure for Extended Inverted File :

Each node at level q of the extended inverted file consists of four fields : *Name* i.e., index field value, *Count* i.e., a field to hold the replication information, and *two pointers (links)*. The links are :

- Classification link (C-link) - Points to the nodes which are at the same level.
- Association link (A-link)- Points to the nodes at level $q+1$ with which the node is associated.

STEPS :

1. Generation of nodes at level 1 :

   For each generalized value of CA_A in $\mathbb{R}_1$, generate a node with *value* field set to generalized value, and *count* field set to the count attribute value. All nodes are linked together using *classification* link (*C-link*). Set *association* link (*A-link*) of each node to NIL.

2. For level $l = 2$ to m do the following :

   For each node i, in level $l - 1$ do the following :

   (a) Use key/CoAs of $\mathbb{R}_{l-1}$ and $\mathbb{R}_l$, find associated values with count attribute value in $\mathbb{R}_l$.

   (b) For each associated value, create a node with *value* field set to it, *count* field set to the count attribute value. Nodes are linked together using *classification* link. Set *association* link to NIL.

   (c) In the list generated, if any two nodes have identical associated value, then one of these nodes is deleted after updating the *count* field of the other node by adding the *count* field value of deleted node to that of the remaining node.

   (d) The association link at node i is made to point to the list at level l.

Figure 4.5 shows the structure of an EIF with depth 'm'.

Explanation : The algorithm **EIFG** leads to the construction of an EIF for our example relations, a part of which is shown for $Sal = 1$, in Figure 4.6.

Note that the EIF represents the associations in a compact form with the help of the *Count* field. EIF differs from the conventional inverted file [Trembley et.al, 76] in two ways :

1. Inverted file is used for indexing in a single relation; in the current context EIF is capturing associations across relations.

Figure 4.5 EIF structure

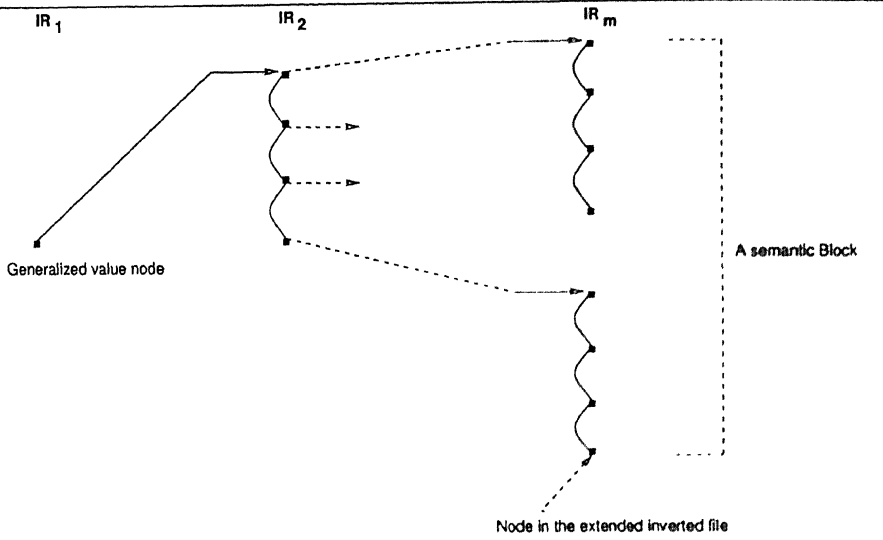
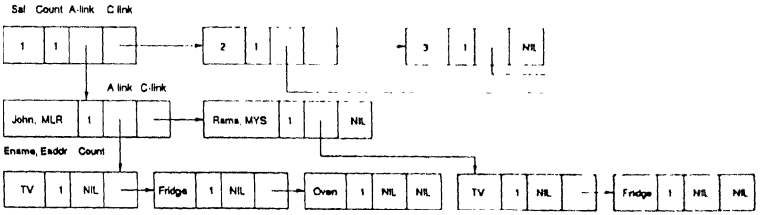


Figure 4.6 EIF for the example



2. EIF is a compact version of inverted file due to the usage of *Count* field.

Similarly, the advantage of using the proposed approach based on EIF structure over the conventional SQL based join-queries [Elmasri et.al, 00] is that

1. in EIF no duplicate information is stored; whereas in the case of SQL-based queries, the resultant relation has duplicate values if the join operation is based on non-key fields.

2. due to the usage of *Count* field, EIF structure consumes less space than the resultant relation obtained using SQL-based join operations.

III-C. Creation of Semantic Partition (CSP) :

INPUT : Extended Inverted file.

OUTPUT : Blocks $B_1, B_2, \ldots, B_\rho$ in the semantic partition of $\mathbb{R}_m$.

STEPS :

For each node i, $(i = 1, 2, \ldots, \rho)$, at level 1 do the following :

1. Use its association link to trace the nodes at level 2.
2. Let $j = 2$, While $(j \leq m)$ do the following :

   Find the set of nodes at level j, S_j, by traversing the nodes at each level, from the selected nodes at level 2 onwards using *characteristic* link to move in the same level and *association* link to move to the next level.
3. The *value* field of each node n, in S_m is repeated X times, where X is given by multiplying all *count* field values of the nodes along the navigation path, from node i, at level 1 to the node n at level m using the association link.
4. The expanded S_m gives semantic partition block, B_i, with respect to the generalized value in the *value* field of node i.

Explanation : Using the algorithm **CSP** the semantic partition block thus created for $Sal = 1$ is shown in Figure 4.7. The number of semantic partition blocks generated

is based on the number of generalized values of the characteristic attribute, CA_A. In our example, the total number of semantic blocks created is three, since there are three generalized values for attribute *Sal*.

Figure 4.7 Semantic partition block for $Sal = 1$

TV
Fridge
Oven
TV
Fridge

IV. Generation of Frequent Tuples and Association Rules (GFTAR):

INPUT : Blocks $B_1, B_2, \ldots, B_\rho$ in the semantic partition of $\mathbb{R}_m$, minimum support, σ and minimum confidence c.

OUTPUT : Frequent tuples in each block B_i and association rules.

STEPS :

For block $i = 1$ to ρ do the following :

1. Apply *Apriori Gen* algorithm [Agrawal et.al, 93B] on B_i to generate frequent tuples. Maintain the count for each frequent tuple generated. Here, the user defined minimum support, σ is used as *minimum support* for the block i. This is because unlike the partition method [Savasere et.al, 95], here the locally frequent tuples themselves are the globally frequent tuples. So, there is no need for the merging phase and an additional database scan as in [Savasere et.al, 95].
2. For each frequent tuple, j with support value α_j, if $(\alpha_j / CA\_COUNT(i)) \geq c$ then output the association rule, $i \Longrightarrow j$. [Here we assumed that there is a one-to-one correspondence between block number and the generalized value of CA_A for simplicity.]

Explanation : Using the algorithm **GFTAR** for the semantic partition block shown in Figure 4.7, and $\sigma = 0.5$, the frequent tuples generated are <TV> and <Fridge>.

With $c = 50\%$, we get the association rules, $(Sal{=}1) \Longrightarrow (Purchased\_Goods{=}\text{TV})$ and $(Sal{=}1) \Longrightarrow (Purchased\_Goods{=}\text{Fridge})$.

4.4 Properties of the Semantic Partition Scheme

In this section we give the proof for soundness and completeness of the semantic partition method. Since the partition algorithm [Savasere et.al, 95] is one of the data-driven algorithms which generates all large itemsets for a given transaction database, we used it as a reference for proving the soundness and completeness properties of the semantic partition method.

4.4.1 Soundness

The large itemsets generated by the semantic partition method are elements of the set of large itemsets generated by the partition algorithm [Savasere et.al, 95] (**Soundness**). (In this section, we used term 'itemset' for 'tuple' to be consistent with the terminology of the partition algorithm discussed in [Savasere et.al, 95].)

Proof Let $S_1, S_2, \ldots, S_n$ be the blocks generated using the semantic partition method and $P_1, P_2, \ldots, P_n$ be the blocks generated using the partition algorithm for the database $\mathcal{D}$ with $\bigcup_i S_i = \bigcup_i P_i = \mathcal{D}$ and $S_i \cap S_j = \emptyset$ for any $i \neq j$, $P_i \cap P_j = \emptyset$ for any $i \neq j$.

Let σ be the user specified support. $Support(x)$ is the fraction of transactions in $\mathcal{D}$ containing x.

In the case of semantic partitioning, we group the items in each block, based on the generalized value of the characteristic attribute and with respect to which we have to generate the association rules. So we have all the itemsets related to a generalized value of the characteristic attribute, CA_A, in each block. So a semantic block contains large itemsets if any, w.r.t a generalized value of CA_A. The proof consists of two parts.

1. To prove that all the itemsets generated by the semantic partition method are present in the set of itemsets generated by the partition algorithm.
2. To prove that one or more itemsets generated by the partition algorithm need not be generated by the semantic partition method.

1. Let $l_1, l_2, \ldots, l_n$ be the set of itemsets generated by the partition algorithm, and $s_1, s_2, \ldots, s_n$ be the set of itemsets generated by the semantic partition method. Since the semantic partition method generates the set of itemsets $s_1, s_2, \ldots, s_n$, they should have support $\geq \sigma$.

   On the other hand, the partition algorithm generates all the itemsets which have the support $\geq \sigma$. So all the itemsets generated by the semantic partition method should be generated by the partition algorithm.

2. To qualify as locally frequent large itemsets in the partition algorithm, each itemset should have support $\geq \sigma/n$, where n is the number of blocks. To qualify as large itemsets in the semantic partition method, they should have support $\geq \sigma$. So locally frequent large itemsets are generated more in the partition algorithm. They may become globally frequent, since in Phase II we scan across the partition (inter-partition scan) and check whether the total support is $\geq \sigma$. Because of this, some itemsets generated in the partition algorithm need not be present in the set of itemsets generated by the semantic partition method.

4.4.2 Completeness

All valid association rules (w.r.t characteristic attribute), generated by the partition algorithm are also generated by the semantic partition method (**Completeness**).

Proof Let $a_1, a_2, \ldots, a_n, b_1, b_2, \ldots, b_n, i_1, i_2, \ldots, i_m, j_1, j_2, \ldots, j_m$ be the set of itemsets such that $a_1 \longrightarrow b_1$, $a_2 \longrightarrow b_2$, $\ldots$, $i_1 \longrightarrow j_1$, $i_2 \longrightarrow j_2$, $\ldots$, $i_m \longrightarrow j_m$, $\cdots$, $a_n \longrightarrow b_n$ be the set of association rules generated by the partition algorithm. Let $i_1, i_2, \ldots, i_m$ be the generalized values of CA_A w.r.t the semantic partition method.

We have to prove that $i_1 \longrightarrow j_1$, $i_2 \longrightarrow j_2$, ... , $i_m \longrightarrow j_m$ are also generated by the semantic partition method.

In the partition algorithm, $i_1 \longrightarrow j_1$, $i_2 \longrightarrow j_2$, ... , $i_m \longrightarrow j_m$ are generated if they are frequent in any of the partitions (this is from the rule that, *the itemset which is globally frequent should be frequent locally*). In the semantic partition method, each partition has all the itemsets corresponding to a generalized value say i_i of CA_A. So they should be frequent, and hence the proof.

4.5 Theoretical Study

4.5.1 Disk Storage Space and Disk Access Time Requirements for EIF

Let $\Re_1, \Re_2, \ldots, \Re_m$ be m relations to be linked. Each $\Re_i$ has x_i number of attributes to be considered. An attribute is said to be considered, if it is a characteristic attribute (CA), or it is an element of key or it is an element of common set of attributes (CoAs) to link to next relation.

Let d_{i1} be the number of nodes in T_A at level l, where T_A is the generalization tree of characteristic attribute. Let $d_{i2}, d_{i3}, \ldots, d_{ix_i}$ be the cardinalities of the domains of other attributes in $\Re_i$. Note that instead of cardinalities of the domain, we can use the number of nodes at a given level, if the generalization tree and level values for the non characteristic attributes in $\Re_i$ are given. Once $\Re_i$ is generalized to $\mathbb{R}_i$, the maximum number of tuples it has, is given by

$$nd_i = d_{i1} \times d_{i2} \times \cdots \times d_{ix_i}$$

The number of attributes in a relation $\mathbb{R}_i$ is $x_i + 1$, where $x_i + 1^{th}$ attribute is the **count** attribute. Initially its value is 1 for each tuple put in $\mathbb{R}_i$ from $\Re_i$ and, whenever we encounter a tuple in $\Re_i$, which already exists in $\mathbb{R}_i$, then we increment the count value of existing tuple and no new entry is made in the table $\mathbb{R}_i$.

Let $nd_1, nd_2, \ldots, nd_m$ be number of tuples in $\mathbb{R}_1, \mathbb{R}_2, \ldots, \mathbb{R}_m$, respectively .

Consider first, $\mathbb{R}_1$ and $\mathbb{R}_2$. Let A_{1j} and A_{2j} be mapping attributes. They are either key or common set of attributes. So for each entry of $\left(\frac{nd_1}{d_{1j}}\right)$ we have at most $\left(\frac{nd_2}{d_{2j}}\right)$ entries, where d_{1j} and d_{2j} are cardinalities of the domains of A_{1j} and A_{2j} respectively. Let $\mathbb{R}_{12}$ be the list (index) thus generated.

Now consider, $\mathbb{R}_{12}$ and $\mathbb{R}_3$. Let A_{12j} and A_{3j} be mapping attributes. For each entry of $\left(\frac{nd_1}{d_{1j}} \times \frac{nd_2}{d_{2j}}\right)$ of $\mathbb{R}_{12}$ we have at most $\left(\frac{nd_3}{d_{3j}}\right)$ entries. Again, d_{3j} is the cardinalities of the domain of A_{3j}. This leads to $\mathbb{R}_{123}$. i.e., index up to $\mathbb{R}_3$ linkage.

In general, for $\mathbb{R}_{12\ldots m}$ we have

$$\frac{nd_1}{d_{1i}} + \frac{nd_1}{d_{1j}} \times \frac{nd_2}{d_{2j}} + \frac{nd_1}{d_{1j}} \times \frac{nd_2}{d_{2j}} \times \frac{nd_3}{d_{3j}} + \quad \text{up to m terms.}$$

$$= \frac{nd_1}{d_{1j}}\left(1 + \frac{nd_2}{d_{2j}}\left(1 + \frac{nd_3}{d_{3j}} \quad 1 + \frac{nd_m}{d_{mj}}\right.\right. \quad \text{number of nodes.}$$

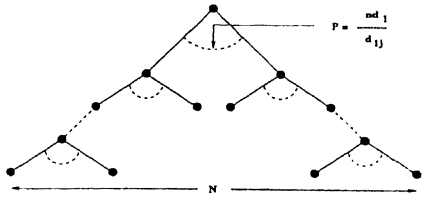
$= \mathbb{N}$.

4.5.1.1 Data Structures

Each node at level q of the extended inverted file consists of four fields : *Name* i.e., index field value - Size X bytes, *Count* - Size Y bytes, and *Two pointers (links)* each with size Z bytes. So, total disk space requirement $= \mathbb{N} \times (X + Y + 2 \times Z)$ Bytes

Diagrammatically, it is shown in figure 4.8. Here m, depth of the linkage, is height of the tree. N is number of leaf nodes which is equal to the number of tuples in a relation which constitute the resultant database. So,

Figure 4.8 A representation of extended inverted file showing number of leaf nodes



$$N = \frac{nd_1}{d_{1j}} \times \frac{nd_2}{d_{2j}} \times \cdots \times \frac{nd_m}{d_{mj}}$$

$$= \prod_{i=1}^{m} \frac{nd_i}{d_{ij}}$$

$$\text{Number of blocks in the semantic partition} = \frac{nd_1}{d_{1j}}$$

The proof for total number of nodes generated in the linking process using mathematical induction is given below :

1. Consider first two relations $\mathbb{R}_1$ and $\mathbb{R}_2$

   Let number of tuples in $\mathbb{R}_1$ be nd_1 and number of tuples in $\mathbb{R}_2$ be nd_2.

   Assume that A_{1i} appearing in $\mathbb{R}_1$ and A_{2i} appearing in $\mathbb{R}_2$ are mapping attributes. Let d_{1i} and d_{2i} be the cardinalities of domains of A_{1i} and A_{2i} respectively. Following are the possible types of mappings between A_{1i} and A_{2i} :-

   (a) 1 : 1 - Primary key Vs. Primary key.

   (b) 1 : N - Primary key Vs. Foreign key.

   (c) N : 1 - Foreign key Vs. Primary key.

   (d) M : N - Foreign key Vs. Foreign key, or Common set of attributes Vs. Common set of attributes.

   If the mapping is 1 : 1, each entry of $\left(\frac{nd_1}{d_{1i}}\right)$ maps to one entry or none of $\mathbb{R}_2$.

   If the mapping is 1 : N, each entry of $\left(\frac{nd_1}{d_{1i}}\right)$ maps to a maximum of $\left(\frac{nd_2}{d_{2i}}\right)$ number of entries of $\mathbb{R}_2$.

   If the mapping is N : 1, each entry of $\left(\frac{nd_1}{d_{1i}}\right)$ maps to a maximum of d_{2i} number of entries of $\mathbb{R}_2$.

If the mapping is $M : N$, each entry of $\left(\frac{nd_1}{d_{1i}}\right)$ maps to a maximum of $\left(\frac{nd_2}{d_{2i}}\right)$ number of entries of $\mathbb{R}_2$.

So in the worst case, each entry of $\left(\frac{nd_1}{d_{1i}}\right)$ maps to a maximum of $\left(\frac{nd_2}{d_{2i}}\right)$ number of entries of $\mathbb{R}_2$.

So, total number of nodes in $\mathbb{R}_{12}$ = number of nodes in $\mathbb{R}_{12}$ at level 1 + number of nodes in $\mathbb{R}_{12}$ at level 2.

$$= \frac{nd_1}{d_{1i}} + \left(\frac{nd_1}{d_{1i}} \times \frac{nd_2}{d_{2i}}\right.$$

2. Consider $\mathbb{R}_{1\ldots(n-1)}$ and $\mathbb{R}_n$.

   Total number of nodes in $\mathbb{R}_{1\ldots n}$ = number of nodes in $\mathbb{R}_{1\ldots n}$ at level $(n-1)$ + number of nodes in $\mathbb{R}_{1\ldots n}$ at level n.

$$\frac{nd_1}{d_{1i}} + \left(\frac{nd_1}{d_{1i}} \times \frac{nd_2}{d_{2i}}\right) + \cdots + \left(\frac{nd_1}{d_{1i}} \times \frac{nd_2}{d_{2i}} \times \cdots \times \frac{nd_{n-1}}{d_{(n-1)i}} \times \frac{nd_n}{d_{ni}}\right.$$

3. Consider $\mathbb{R}_{1\ldots n}$ and $\mathbb{R}_{(n+1)}$.

   Total number of nodes in $\mathbb{R}_{1\ldots n+1}$ = number of nodes in $\mathbb{R}_{1\ldots n+1}$ at level n + number of nodes in $\mathbb{R}_{1\ldots n+1}$ at level $(n+1)$.

$$= \frac{nd_1}{d_{1i}} + \left(\frac{nd_1}{d_{1i}} \times \frac{nd_2}{d_{2i}}\right) + \cdots + \left(\frac{nd_1}{d_{1i}} \times \cdots \times \frac{nd_n}{d_{ni}}\right) + \left(\frac{nd_1}{d_{1i}} \times \cdots \times \frac{nd_n}{d_{ni}} \times \frac{nd_{n+1}}{d_{(n+1)i}}\right.$$

So, the total number of nodes in the extended inverted file, $\mathbb{R}_{1\ldots n}$ is

$$\frac{nd_1}{d_{1i}} + \left(\frac{nd_1}{d_{1i}} \times \frac{nd_2}{d_{2i}}\right) + \cdots + \left(\frac{nd_1}{d_{1i}} \times \cdots \times \frac{nd_n}{d_{ni}}\right.$$

Hence the number of leaf nodes in the extended inverted file is

$$\frac{nd_1}{d_{1i}} \times \cdots \times \frac{nd_n}{d_{ni}}$$

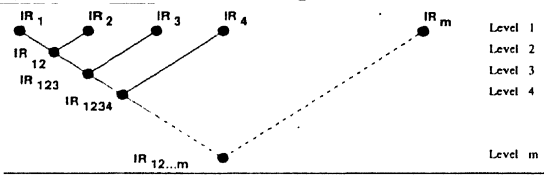
4.5.1.2 Indexing time

Here, we calculate the total disk access time required to do the indexing. It is given by the product of the total number of disk accesses and the access time, where access time is a constant. The total number of disk accesses is obtained as follows.

Let $\mathbb{R}_1, \mathbb{R}_2, \ldots, \mathbb{R}_m$ be m relations. We used binary ladder type of linking procedure [Elmasri et.al, 00] to construct the extended inverted file.

Binary ladder type of linking Figure 4.9 shows this approach of linking. Here $\mathbb{R}_{12\ldots i}$ indicates index that is generated at i^{th} level, where $2 \leq i \leq m$. Since, we can get number of nodes generated at each level, and we know the size of each node, we can determine the number of disk blocks for indexing at each level.

Figure 4.9 Binary ladder type of linking



Assumptions : Following assumptions are made for calculating the number of disk accesses.

1. nB is the number of buffers available and buffer size = disk block size = B.
2. Disk blocks for a relation are packed. i.e., each disk block has the tuples of only one relation. And disk blocks of a relation are linked.
3. Blocking factor, i.e., number of records per disk block is given by $bfr = \lfloor \frac{B}{\text{tuple size}} \rfloor$

Let $b\mathbb{R}_{12}, b\mathbb{R}_{123}, \cdots, b\mathbb{R}_{12\ldots q}$ be the number of disk blocks required at level q where q $2, 3, \cdots, m$. We have $bfr = \lfloor \frac{X+Y+2\times Z}{} \rfloor$.

So, the number of disk blocks required at each level i (where $2 \leq i \leq m$) is given by

$$b\mathbb{R}_{12\ldots i} = \left\lceil \frac{\#\text{ of nodes generated at level } i}{bfr} \right\rceil$$

Using the block oriented iteration method, and considering the smallest relation outside [Elmasri et.al, 00], the total number of disk accesses is given by

$$\sum_{q=1}^{m-1}\left[\text{Smaller}\left(b\mathbb{R}_{12\cdots q}, b\mathbb{R}_{q+1}\right) + \left\lceil\frac{\text{Smaller}\left(b\mathbb{R}_{12\cdots q}, b\mathbb{R}_{q+1}\right)}{nB-1}\right\rceil \times \text{Larger}\left(b\mathbb{R}_{12\cdots q}, b\mathbb{R}_{q+1}\right) + b\mathbb{R}_{12\cdots q+1}\right] \tag{4.1}$$

Here, $Smaller(a,b)$ function returns the smaller value of a and b and $Larger(a,b)$ returns the larger value of a and b.

The term $b\mathbb{R}_{12\cdots q+1}$ in the expression (1) is for writing the index blocks generated at $(q+1)^{th}$ level.

In order notation, the expression (1) reduces to $\bigcirc\left(\frac{mb\mathbb{R}^2}{nB}\right)$ which is $\bigcirc\left(\frac{mN^2}{nB}\right)$, since $= \frac{N}{\text{Constant}}$.

4.6 Experimental Results

We used three databases for the experimental study. One is PERSON database, having attributes *Salary*, *Age*, *SSN*, *Education* and *Number_of_vehicles*. Descriptions of these attributes are :

Salary - Uniformly distributed between 5,000 and 30,000.

Age - Uniformly distributed between 20 and 60.

SSN - Social Security Number, a unique identifier of the person.

Education - Uniformly chosen between 0 and 4 corresponding to different levels of education.

Number_of_vehicles - Uniformly chosen between 0 and 5.

The second database is, SECURITY_INFO database, having two attributes *Sec_no* and *CCN*. Descriptions of these attributes are :

Sec_no - Security number, unique identifier of a person.

CCN - Credit card number. This number is unique to a person; but a person can have more such cards. Number of *CCN* per *Sec_no* is uniformly chosen from 1 to 6.

SSN and *Sec_no* are Common set of attributes (CoA) by definition.

Both PERSON and SECURITY_INFO databases are generated randomly with Common set of Attributes (CoAs) being equivalent. We generated 100K tuples for each databases. All random numbers generated are integers.

The third database is, PURCHASE database, having attributes *Card_no* and $Goods\_purchased_i$, $0 \leq i \leq 999$. Descriptions of these attributes are :

Card_no - Credit card number. It is generated randomly but with the same domain as *CCN*.

$Goods\_purchased_i$, $0 \leq i \leq 999$ - Each goods purchased, $Goods\_purchased_i$ is having domain {0,1} indicating either their absence or presence in the buying patterns of a customer. The buying patterns of customers (i.e., the values for $Goods\_purchased_i$) are generated based on the synthetic data generator given in [Quest site]. So, totally there are 1001 attributes in PURCHASE database. The parameters used in the generation of the synthetic data are shown in table 4.1. Table 4.2 shows the names and parameter settings for the data sets used.

CCN and *Card_no* are Common set of attributes (CoA) by definition.

Table 4.1 Parameters

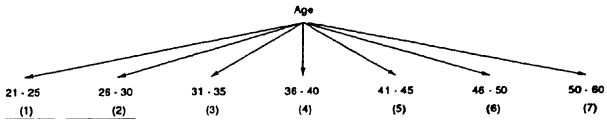
$\|D\|$	Number of tuples
$\|T\|$	Average size of tuples
	Average size of maximal potentially frequent tuples
	Number of maximal potential frequent tuples
NOI	Number of attributes each having domain {0,1}

Table 4.2 Parameter settings

Name	$\|D\|$	$\|T\|$	$\|I\|$	$\|L\|$	NOI
SET-1	100K	10	4	1000	1000
SET-2	100K	10	4	2000	1000

We want to find the associations between the values of attributes *Age* and tuples due to 1000 attributes, $Goods\_purchased_i$, $0 \leq i \leq 999$. The generalization tree corresponding to *Age* used in our experiment is shown in Figure 4.10. Our algorithm, **AMAAM** generates

Figure 4.10 Generalization tree for *Age*



the navigation path across the databases using CoAs, i.e., *SSN* and *Sec_no* between PERSON and SECURITY_INFO; *CCN* and *Card_no* between SECURITY_INFO and PURCHASE and is pictorially shown in Figure 4.11.

Figure 4.11 Navigation path generated by **AMAAM**

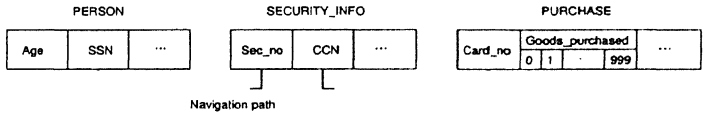


Table 4.3 shows the seven blocks of the semantic partition generated by **AMAAM** (since '*Age*' is generalized in to seven values) for both sets of data, i.e., SET-1 and SET-2 with different support values, i.e., with 0.25% and 0.90%.

Description of the entries in table 4.3 :

No. of FTs : This entry gives a total number of *k-frequent tuples* per block, where $k = 1, 2, 3, \ldots, NOI$. Here k-tuples are instantiations of Cartesian product of corresponding k attributes.

COMP : This entry gives how many times we accessed the block, to check whether the tuple generated is having required *minimum support* to qualify as a frequent tuple or not.

Size : This entry gives the size of the block in PURCHASE database corresponding to each generalized value *Age*. In the case of semantic partition, size of the block is number of associations on the PURCHASE database for each generalized value of the characteristic attribute, *Age*.

No. of Assoc. Rules : This entry gives the total number of association rules generated with the confidence, $c = 5\%$.

Table 4.3 No. of FTs and association rules generated for semantic partition blocks

Data Set	Min. Sup. (σ)	No.of FTs	No.of Assoc. Rules (c=5%)	Size	COMP	Block No.
SET-1	0.25%	206	18	14004	19112	1
		211	17	14134	21322	2
		206	19	14074	20101	3
		199	16	13972	18722	4
		203	18	14035	19307	5
		201	18	13902	18722	6
		211	17	14115	19903	7
	0.90%	5	5	14004	10	1
		6	6	14134	15	2
		5	5	14074	10	3
		5	5	13972	10	4
		5	5	14035	10	5
		7	7	13902	21	6
		5	5	14115	10	7
SET-2	0.25%	199	13	14027	19701	1
		191	12	14154	18145	2
		200	12	14112	19900	3
		191	13	14000	18145	4
		188	14	14059	17578	5
		186	13	13923	17205	6
		195	14	14141	18915	7
	0.90%	3	3	14027	3	1
		2	2	14154	1	2
		3	3	14112	3	3
		3	3	14000	3	4
		3	3	14059	3	5
		3	3	13923	3	6
		2	2	14141	1	7

From table 4.3, we can make the following observations.

In table 4.3, the entries under 'No. of FTs', correspond to the frequent tuples which are globally frequent tuples. This is because all frequent tuples for a generalized value of the characteristic attribute lie in a single semantic block. A tuple (tp) is said to be frequent, if support(tp) $\geq \sigma$. The number of frequent tuples decreases with increase in σ

value. This is shown in Table 4.3 under the column 'No. of FTs' between $\sigma = 0.25\%$ and $\sigma = 0.90\%$ value.

Along with the FTs, we also maintain their supports. So, from FTs, we can check whether the corresponding association rules are permitted or not based on c. So, there is no more extra processing effort required to generate the rules from the frequent tuples. This is possible because, by specifying attributes between the values of which we want to find the associations, we fix the antecedent side of the association rule.

4.7 Summary

We have presented a scheme which uses a semantic network, generalization trees, and a collection of databases for mining *inter-database association rules*. The scheme builds a navigation path and an extended inverted file structure to link the attributes across the databases and generates associations between generalized values and values of characteristic attributes.

We showed that it is possible to efficiently obtain a semantic partition of the resultant database using a collection of databases and a semantic network. Further, generation of frequent tuples needs scanning the resultant database *only once*, since it is semantically partitioned. The locally-frequent tuples (which are also the globally-frequent tuples) can be used along with their support value to obtain association rules between each value (antecedent) in the generalized values of the characteristic attribute and all frequent tuples (consequent) in each semantic block. So, no more processing effort is required for generating the association rules from the frequent tuples. Since the semantic partition generates the rules between the values of specified attributes, the association rules generated are meaningful.

Chapter 5

And-Or Concept Taxonomies for Mining Generalized Association Rules

5.1 Introduction

In this chapter, we introduce a new approach to mine *relevant* generalized association rules. Here, we propose schemes which use *explicit* knowledge along with the database as input [cf. 3.4.1] for finding associations.

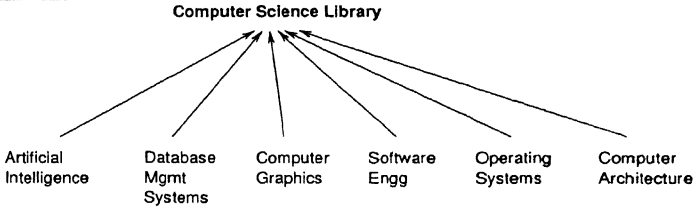
Generalized association rules based on *is_a* hierarchies are introduced in [Srikant et.al, 95]. For example, $Brand\_A$ *is_a* $Soft\_drink$, and from an association rule, $Brand\_A \Longrightarrow Chips$, one may infer a generalized rule, $Soft\_drink \Longrightarrow Chips$.

Mining for negative association rules based on generalization is introduced in [Savasere et.al, 98]. For example, consider a taxonomy where $Brand\_A$ and $Brand\_B$ are two brands of $Soft\_drinks$. After scanning through a large number of transactions, if it is found that, when customers buy $Ruffles$ they also usually buy $Brand\_B$ but not $Brand\_A$, then we can conclude that $Ruffles$ has an interesting negative association rule with $Brand\_A$. The reason for this is that by considering the association between $Ruffles$ and $Brand\_B$, one may expect the association between $Ruffles$ and $Brand\_A$ also to be present because

Brand_A and *Brand_B* are *Soft_drinks*. The negative association rules are not obtained while mining for the positive association rules. So, mining for positive and negative associations is to be carried out separately.

The *is_a* hierarchy based knowledge employs an OR node, where if at least one of the child nodes has sufficient support, then the parent node will also have sufficient support. This is not always adequate. For example, consider an *is_a* hierarchy corresponding to *Computer Science Library (CSL)* based on computer books shown in Figure 5.1. It is difficult to support the library as *CSL* if it has *only* good collection of *Artificial Intelligence* books but it does not have sufficient collections in other areas. Also note that there are

Figure 5.1 Is_a taxonomy for Computer Science Library (CSL)



other types of relationships like *is_part_of* which can be captured using AND nodes. So, to capture all the generalized knowledge in a flexible manner, we introduce a new taxonomy called AND-OR taxonomy (or simply, A O taxonomy). This taxonomy consists of both AND nodes and OR nodes.

In this chapter, we introduce a knowledge-based scheme for generalization of items. We also introduce an extended version of positive association rules which can capture the semantics of negative association rules of the form discussed in [Savasere et.al, 98]. We mine for associations between items/group of items which are not typically expected by the user and are also not obvious from the knowledge base. We also claim that the mining is relevant because we find the rules among the items/concepts within the A O taxonomy provided by the user.

The contributions presented in this chapter are

- *Introducing a new taxonomy, called A O taxonomy, for knowledge representation;*

so far only 'is_a' taxonomy is used in the literature for association rule mining.

- *Using the A O taxonomy for* ***relevant*** *mining of association rules;* this is different from the conventional scheme of mining all possible association rules.

- *Developing a new and novel algorithm called 'A O Single scan algorithm' that discovers interesting association rules using only* ***one database scan.***

- *Providing a direct mechanism of discovering generalized, positive and negative association rules.*

The rest of this chapter is organized as follows. Section 5.2 gives the problem formulation. In this section, we define the problem statement and state the motivation for this work. We also introduce the notion of A O taxonomy. Generation of generalized association rules based on the notion of A O taxonomy is given in section 5.3. We discuss the generation of interesting association rules using the scheme, which needs a single database scan, in section 5.4. Experimental results are discussed in section 5.5. In section 5.6, we summarize our study.

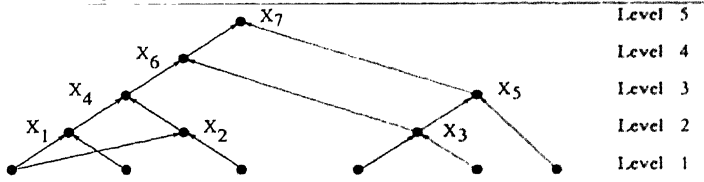
5.2 Problem Definition

In this work, we concentrate on generating generalized (including negative) association rules corresponding to the values of attributes which belong to the relations of one or more databases in set $\mathbb{D}$. For the sake of simplicity and without loss of generality, we assume that the relation of reference (which is valuetable without the constraint specification) already exists and further, it is a transaction database, $\mathfrak{D}$. So, in this chapter, we use the terms 'item' and 'itemset' instead of 'value' and 'tuple' respectively to be consistent with the literature. We use a knowledge base in the form of an A O taxonomy, a kind of semantic network. We propose - complement of support, which is required for natural mining of negative association rules. We introduce a special type of negative association rule of the form $A \Longrightarrow \neg B$, where A and B are items or concepts.

5.2.1 A O Taxonomy

A taxonomy is a mapping from a set of child nodes at some level $l - i$ (i is such that $1 \leq (l - i) < l$), to their parent node at level l. The nodes at level $l = 1$ are leaf nodes and are called *item nodes*. For example, in Figure 5.2, $x_1, x_2, \ldots, x_6$ represent items. Each node at level $l > 1$ is a non-leaf node and is called a *concept node*. For example, in Figure 5.2, $X_1, X_2, X_3, \ldots, X_7$ represent concepts at levels 2 and above.

Figure 5.2 A taxonomy



We define the *A O* taxonomy [1] as a taxonomy made up of AND-nodes and OR-nodes. A Characteristic of the A O taxonomy is that it is a DAG with single root node. We define the AND taxonomy and OR taxonomy below.

AND-node : A concept node which has enough support if *all its child nodes simultaneously* have support $\geq \sigma$, where σ is a user defined minimum support.

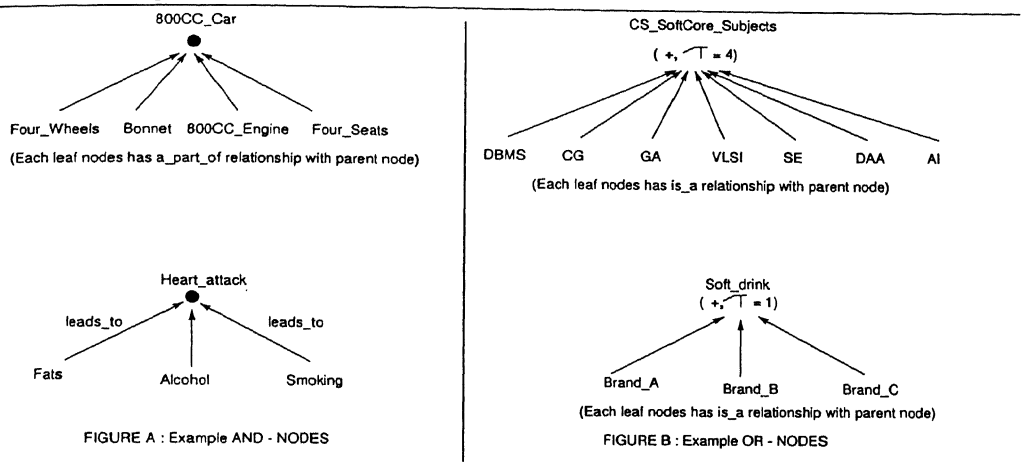
Here, *simultaneous* means - the co-occurrence of items directly, if the child node is an item or indirectly, if the child node is a concept in a tuple. Figure 5.3A shows example of AND-nodes. We use a $\bullet$ to represent an AND node.

OR-node : A concept node which has enough support if *atleast τ of its child nodes simultaneously* have support $\geq \sigma$, where σ is a user defined minimum support.

Let these τ nodes be $N_1, N_2, \ldots, N_\tau$. Note that $\{N_1, N_2, \ldots, N_\tau\}$ is a frequent τ-itemset/conceptset.

[1] It is functionally different from AND-OR graph, which is used in search strategies in AI context. We used the term A O taxonomy because its nodes resemble AND and OR functions in some sense.

Figure 5.3 Example taxonomies



Basically, this node captures the generalized version of an item/concept in a restricted manner. We call τ the OR-threshold. Figure 5.3B shows example OR-nodes. We use $(+,\tau = c)$ to represent an OR node with a value of τ equal to c. Note that if τ value of an OR node is equal to the number of its child nodes, then the OR node is effectively equivalent to an AND node.

In the work reported in [Han et.al, 95; Han and Fu 99], the authors use an is_a hierarchy for mining multi-level associations. They use the following property of the is_a hierarchy. If a concept is frequent, only then its descendents (items/concepts) are to be explored for possible associations. We can also use the same strategy in the case of A O taxonomies, if we have only OR nodes with the value of $\tau = 1$. For example, in Figure 5.3B, if *Soft_Drink* is infrequent, then none of its descendents is frequent. On the other hand, for AND nodes, and OR nodes with $\tau > 1$, even if the parent node is infrequent some of its descendents may be frequent. For example, in figure 5.3A, even if the concept *800CC_Car* is infrequent, the descendents like *Four_Wheels* may be frequent. So based on the situation at hand, we have to explicitly check the frequency of items/concepts which are descendents of other concepts. So, we cannot directly use the simplification suggested in [Han et.al, 95; Han and Fu 99].

Note that in the A O taxonomy, an item/concept may appear in a complemented or

a non-complemented form. Complement of a leaf node (i.e., item at level 1) denoted as $\neg x$, is supported if the number of transactions in which the item is absent is more than or equal to the user defined minimum support.

Complement of a concept (i.e., a node at level l other than 1), denoted as $\neg X$ is supported as defined below.

1. If the concept at level l corresponds to an AND node, then complement of *at least one* child node (i.e., item/concept) has support more than or equal to the minimum support.

2. If the concept at level l corresponds to an OR node, then *at most* τ - 1 of its child nodes (i.e., items/concepts) have support more than or equal to the minimum support.

   Note that the above two definitions are recursive with respect to levels.

Illustrative example for AND-OR taxonomy

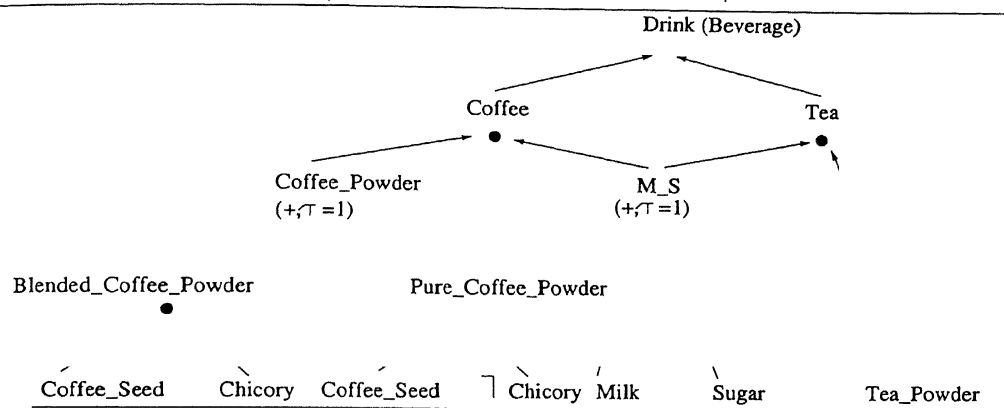
Figure 5.4 shows an AND-OR taxonomy. In this taxonomy, the concept *Pure_Coffee_Powder* gets enough support if we have sufficient support for both *Coffee_Seed* and $\neg$*Chicory*. Note that if $\neg$*Chicory* does not have enough support, then *Pure_Coffee_Powder* may not get supported. The concept *Coffee_Powder* gets enough support if we have sufficient support for at least one of *Blended_Coffee_Powder* and *Pure_Coffee_Powder*.

5.2.2 Complement of Support ($\tilde{S}$upport)

We know that Support(ϑ) is the fraction of transactions that contain ϑ, where ϑ is an item or a concept. We define S̃upport of an item, x as follows. x has a s̃upport s, in the transaction database $\mathfrak{D}$, if s% of the transactions in $\mathfrak{D}$ do not contain x. In other words, it is the ratio of total number of transactions which have a value '0' for x in the transaction database, $\mathfrak{D}$, to the total number of transactions in $\mathfrak{D}$.

S̃upport of a concept X can be defined recursively over all of its child nodes. Each of the child nodes is either a concept or an item. We represent X, whose s̃upport should be

Figure 5.4 AND-OR taxonomy



used to find the frequent itemset, by $\neg X$ in A O taxonomy. Complement of a concept is already explained in section 5.2.1. So, $\tilde{\text{S}}\text{upport}(X) \equiv \text{Support}(\neg X)$.

5.2.3 AND-OR Generalization Rule

It consists of AND generalization rule and OR generalization rule.

AND Generalization Rule : For an AND node at level l, the generalization to the corresponding concept, at level l, is done if all of its child nodes at level $l - i$ (where i is such that $1 \leq (l - i) < l$) simultaneously have support $\geq \sigma$, a user specified support.

τ-OR Generalization Rule : For an OR node at level l, the generalization to the corresponding concept, at level l, is done if at least τ of its child nodes at level $l - i$ (where i is such that $1 \leq (l - i) < l$) simultaneously have support $\geq \sigma$, a user specified support.

Example : In Figure 5.4, we can generalize to the concept $Coffee$ if $Coffee\_Powder$ and any one (at least) of $Milk$ or $Sugar$ have support $\geq \sigma$. Similarly, we can generalize to the concept $Pure\_Coffee\_Powder$ if Support($Coffee\_Seed$ and $\neg Chicory$) $\geq \sigma$.

5.2.4 Motivation for AND-OR Generalization

As discussed in the introduction, the currently available mining algorithms for generating generalized or negative association rules are based on *is_a* taxonomies and as a consequence have the following limitations :

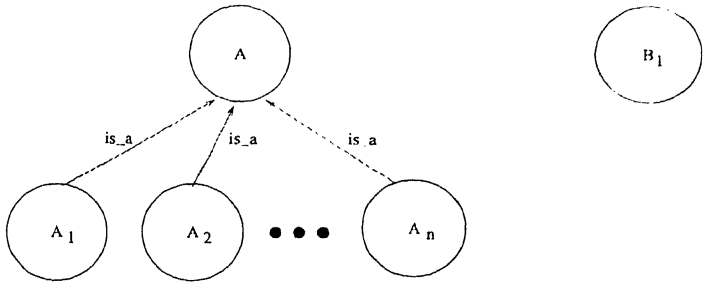
- The *is_a* taxonomy used does not capture enough semantics to handle the type of generalization the user is looking for.

  For example, the generalized item like **Computer** cannot be captured, even when all the components of a Computer like *Mother-board*, *Keyboard*, *Cabinet*, *Hard-disk*, $3\frac{1}{2}$" *Floppy disk drive*, *Mouse*, *CD-ROM drive* and *Monitor* are items that co-occur in a large number of transactions. This type of generalization is very much useful to find higher level association rules like *Computer* $\Longrightarrow$ *Air-conditioner* or *Computer* $\Longrightarrow$ *Shade-light* when the *Computer* components are bought along with parts of *Air-conditioner* or when the *Computer* components are bought along with parts of *Shade-light*. In other words, there is a need to generalize based on *a_part_of* relationship (functional relationship).

- Generalized association rules generated based on the *is_a* taxonomy may have many exceptions which is counter-intuitive.

  We can characterize this behaviour using an *is_a* taxonomy given in Figure 5.5. In

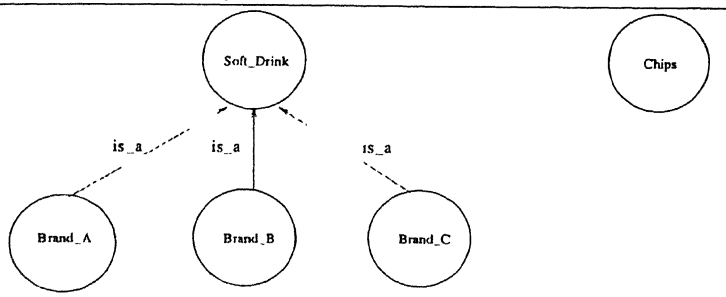
Figure 5.5 Is_a taxonomy



this figure, let A be the ancestor of items $(A_1, A_2, \ldots, A_n)$ and let B_1 be another item. There exist an association rule, $A \Longrightarrow B_1$ if one or more of $A_i \Longrightarrow B_1$ is true,

where $1 \leq i \leq n$. For example, $A_1 \Longrightarrow B_1$ holds, but none of $A_i \Longrightarrow B_1$ $(2 \leq i \leq n)$ may hold. This leads to a rule $A \Longrightarrow B_1$ with many exceptions, since only few of A_is have association with B_1. As a specific example, consider the is_a taxonomy given in Figure 5.6. Generalization to *Soft_drink* is done from items *Brand_A*, *Brand_B*

Figure 5.6 Example is_a taxonomy



or *Brand_C*. In other words, if *Brand_A* (but not *Brand_B* or *Brand_C*) has enough support, then *Soft_Drink* is assumed to have enough support. As a consequence, an association rule of the form $Soft\_Drink \Longrightarrow Chips$ could be generated leading to the exceptions like $Brand\_B \not\Longrightarrow Chips$ and $Brand\_C \not\Longrightarrow Chips$. So, it is possible to have many exceptions to the same rule.

- Over generalization.

  Each of the child nodes in the *is_a* taxonomy is replaced by its immediate parent node. In such a situation, even though none of the child nodes is having enough support, the parent node may have the required support. In Figure 5.5, it may so happen that there exists an association rule, $A \Longrightarrow B_1$ even though none of A_is $(1 \leq i \leq n)$ has association with B_1. As a consequence, the association rule in Figure 5.6, $Soft\_Drink \Longrightarrow Chips$ is generated even though $Brand\_A \Longrightarrow Chips$, $Brand\_B \Longrightarrow Chips$ and $Brand\_C \Longrightarrow Chips$ are not possible. We call this type of generalization *over generalization*. This shows that there is a need to generalize based on a collective support of child nodes.

The limitations mentioned above lead to the motivation for the use of *AND-OR* taxonomy (*A O* taxonomy) for generalization. This taxonomy captures, in a natural way,

generation of concepts which can be derived from lower level concepts/items. Here, generalization to the parent OR node is not automatic even if one of its child nodes has enough support. We permit this generalization in a restricted way with the help of the parameter τ.

5.3 Generation of Association Rules

In this section we describe an algorithm to discover the generalized association rules based on the A O-taxonomy discussed in the previous section. We call this algorithm **A O Regular**. Later in this section, we describe characteristics of the association rules that are generated using **A O Regular**.

A O Regular

Input

1. Transaction database, $\mathfrak{D}$.
2. User defined minimum support, σ, and minimum confidence c.
3. A O-taxonomy, $\mathfrak{T}$, with height, T, and Level of Interestingness, $\mathfrak{M}$ with respect to $\mathfrak{T}$. $\mathfrak{M}$ gives the upper bound on the level till which we have to mine using the A O taxonomy. By changing the value of $\mathfrak{M}$ we can change the level of interestingness. So, because of the parameter $\mathfrak{M}$, we have **interactive mining**. [Note : $1 \leq \mathfrak{M} < T$.]

Output

Generates all possible association rules w.r.t $\mathfrak{T}$.

Steps

I. Generation of frequent items/concepts :

Let $\mathfrak{L}$ be the list to hold the itemsets, which is initially empty.

1. Determine the *descriptions* of all nodes in $\mathfrak{T}$, using **Description Generation Algorithm (DGA)**. **DGA**, for input ϑ (where ϑ is a node or a collection of nodes), outputs all possible combinations of items (i.e., set of one or more sets of items at level 1) on which support of ϑ depends. We give an example to explain this.

   An Example : Consider the AND-OR taxonomy shown in Figure 5.4. **DGA**, for input *Drink*, outputs the list {{*Coffee_Seed Chicory Milk Tea_Powder*} {*Coffee_Seed Chicory Sugar Tea_Powder*} {*Coffee_Seed* ¬*Chicory Milk Tea_Powder*} {*Coffee_Seed* ¬*Chicory Sugar Tea_Powder*}}. Note that components of an AND node are included in *a* set; components of each child node of an OR node are put in a *separate* set depending on the value of τ.

   In this example, the node *Drink*(*Beverage*) has enough support if any one of the sets {*Coffee_Seed Chicory Milk Tea_Powder*}, {*Coffee_Seed Chicory Sugar Tea_Powder*}, {*Coffee_Seed* ¬*Chicory Milk Tea_Powder*} and {*Coffee_Seed* ¬*Chicory Sugar Tea_Powder*} has enough support (i.e., support $\geq \sigma$) w.r.t the database $\mathfrak{D}$. In general, let $\{S_1, S_2, \ldots, S_x\}$ be the description of ϑ, where each S_i, $(i : 1 \ldots x)$ is a set. ϑ is said to have enough support, if *at least* one $S_i \in \mathbf{DGA}(\vartheta)$ has support $\geq \sigma$ w.r.t $\mathfrak{D}$.

2. For each node i of $\mathfrak{T}$ at level, $l = 1$ to $\mathfrak{M}$ do the following :

   - Let $S = \mathbf{DGA}(i)$.
   - If (Support(S_j) $\geq \sigma$ for any $S_j \in S$) then

     Put node i in the list $\mathfrak{L}$.

Explanation : The above steps give the items and concepts which are to be considered for finding the association rules. Note that the list $\mathfrak{L}$ contains heterogeneous elements. They are either items or concepts.

II. Generation of the frequent itemsets/concepts :

1. **Main** Algorithm.

   INPUT : $\mathfrak{L}$.

   OUTPUT : Frequent itemsets in $\mathbb{L}$.

   STEPS :

   Let $\mathbb{L}_1 = \mathfrak{L}$.

   Let $k = 2$, where k represents the iteration number.

   - While $(\mathbb{L}_{k-1} \neq \emptyset)$

     begin

     C_k = **Gen**$(\mathbb{L}_{k-1})$ // **Gen** algorithm is discussed below.

     $\mathbb{L}_k = \{c \in C_k \wedge c.count \geq \sigma\}$

     $k = k + 1$

     $\mathbb{L} = \mathbb{L} \cup \mathbb{L}_k$ // $\mathbb{L}$ is initially empty.

     end.

2. **Gen** Algorithm

   INPUT : $\mathbb{L}_{k-1}$.

   OUTPUT : C_k.

   STEPS :

   - For $i = 1$ to $|\mathbb{L}_{k-1}| - 1$

     begin

     For $j = i + 1$ to $|\mathbb{L}_{k-1}|$

     begin

     $\mathfrak{l}_i$ = **DGA**$(\mathbb{L}_{k-1}[i])$.

     $\mathfrak{l}_j$ = **DGA**$(\mathbb{L}_{k-1}[j])$.

   Note : If $\mathbb{L}_{k-1}[x]$ (where x is either i or j), is a set of nodes in the taxonomy, say $\{N_1, N_2, \ldots, N_r\}$ then find the description for each N_i, $1 \leq i \leq r$. $\mathfrak{l}_x$ is the union of these descriptions.

   Let c.count be the support corresponding to $\mathfrak{l}_i \cup \mathfrak{l}_j$ w.r.t $\mathfrak{D}$.

Note : $\mathfrak{l}_i \cup \mathfrak{l}_j$ is obtained by concatenating each itemset of $\mathfrak{l}_i$ with each itemset of $\mathfrak{l}_j$ and removing duplicate subset of items if any.

$C_k = C_k \cup \{\mathbb{L}_{k-1}[i] \cup \mathbb{L}_{k-1}[j]\}$ // C_k is initially empty.

end

end

Explanation : The **Main** algorithm generates frequent itemsets/concepts from the candidates generated by **Gen** algorithm. The method used is - generate k-potentially frequent itemsets/concepts from (k-1)-frequent itemsets/concepts. The two steps, **I** and **II** can be performed for each partition, P_i, as in Partition algorithm [Savasere et.al, 95]. Where, $\bigcup_i P_i = \mathfrak{D}$. So, we need at most two database scans.

III. Generation of Association Rules :

INPUT : $\mathbb{L}$.

OUTPUT : Set of Association rules.

STEPS :

For each entry of $\mathbb{L}$, one has to examine one of the following cases :

Case 1 : All are items, $a_1, a_2, \ldots, a_n$ ($I = a_1, a_2, \ldots, a_n$) belong to level 1 in $\mathfrak{T}$:

- If they are *all* siblings of AND/OR Node then

  If their parent exist in $\mathbb{L}$ then

  Ignore that sibling.

  Else Find the association rules as in [Agrawal et.al, 94] among I.

Case 2 : If some are items $a_1, a_2, \ldots, a_n$, ($I = a_1, a_2, \ldots, a_n$) that belong to level 1 and others are concepts $A_1, A_2, \ldots, A_m$ that belong to level > 1 :

Note : $\mathbb{L}$ does not contain parent-child group as frequent itemsets.

Let $\mathfrak{l}$ be initially empty.

- For each A_i, where $1 \leq i \leq m$

  $\mathfrak{l}_i$ = **DGA**(A_i)

$I = I \cup I_i$

Find the association rules as in [Agrawal et.al, 94] between I and A_i and among A_is by using items in I and descriptions in l.

Case 3: All are concepts, $A_1, A_2, \ldots, A_m$ that belong to level > 1 :

Let I is initially empty.

- For each A_i, where $1 \leq i \leq m$

  $I_i = \mathbf{DGA}(A_i)$

  $I = I \cup I_i$

  Find the association rules as in [Agrawal et.al, 94] between A_is by using their corresponding descriptions available in I.

Explanation : For each entry in $\mathbb{L}$, if the element in $\mathbb{L}$ is a collection of items then generate association rules using that collections as in [Agrawal et.al, 94]. Otherwise, use the descriptions of the concepts which is a collection of items along with other collection of items if any for generation association rules as in [Agrawal et.al, 94].

5.3.1 Characteristics of Association Rules Generated

Following are the characteristics of the association rules, generated using A O taxonomy.

1. Since we are allowing the complement of the items to participate in finding the frequent itemsets, the following type of positive and negative association rules are possible.

   (a) $A \Longrightarrow B$, where each of A and B is either an item, set of items or concepts.

   (b) $A \Longrightarrow \neg B$ and its variants, where each of A and B is either an item or set of items or concepts.

   The first type of rule is equivalent to any positive rule, if A and B are simple items; otherwise, they represent associations between functionally generalized or *restricted is_a* generalized concepts. The second type of rule captures the semantics of negative

association rules of the form $A \not\Longrightarrow B$ of [Savasere et.al, 98]. Note that in our case, generation of negative association rules is natural and is not combinatorially explosive, because in any case, generation of a rule of the form $A \Longrightarrow \neg B$ is possible only if $\neg B$ is present in the A O taxonomy and $\tilde{\text{S}}\text{upport}(B) \geq \sigma$.

2. In an association rule, $A \Longrightarrow B$, we permit $A \cap B$ to be non-empty, whereas in the literature it is tacitly assumed that $A \cap B = \emptyset$. For example, using the A O taxonomy in Figure 5.4, one can have the association rule as $Coffee \Longrightarrow Tea$. Here, items $Milk$ and $Sugar$ are common to both concepts $Coffee$ and Tea.

3. **Relevant association rule mining :** We explore the possibility of generating association rules among items and concepts available in the given taxonomy. Moreover, the grouping of the items is based on semantics and not on any arbitrary consideration. Since the taxonomy is given as input, we find the interesting rules among the items/concepts within this taxonomy only. Such mining is called **relevant association rule mining**.

5.4 Generation of Association Rules using a Single Scan Algorithm

In the last section, we described an algorithm which gives generalized association rules. The algorithm shows a natural way of generalizing the items, using an A O taxonomy. Since we used the conventional way of grouping the items, the algorithm generates a large collection of itemsets and hence a large number of association rules. In this section, we propose an algorithm for discovering only interesting association rules.

Definitions :

2-nodeset : It is the set of the form $\{A, B\}$, where A and B correspond to concepts/items in an A O taxonomy.

Frequent 2-nodeset : A 2-nodeset is said to be *frequent* if it has the support greater than or equal to the user defined minimum support (σ).

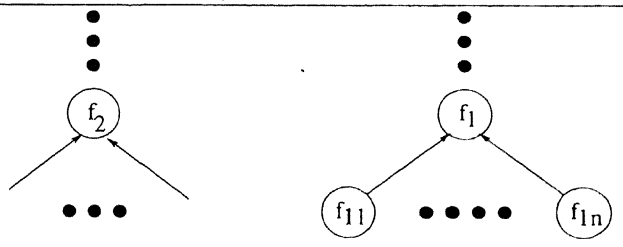
Complete A O taxonomy : An A O taxonomy is said to be complete, if all the required concepts are present.

In this section, we describe an algorithm, **A O Single Scan** which not only generates all the interesting association rules, but also accesses the transaction database **only once**. The algorithm is based on the following lemma.

Lemma 1 In finding the association rules based on a complete A O taxonomy, it is sufficient to find frequent 2-nodesets.

In an A O taxonomy, it can be observed that the nodes at level 2 or above represent the concepts, input by the user. Each of the concepts at any level (≥ 2) is finally based on a collection of items at level 1 of the taxonomy. The upper level concepts are built from the lower level concepts. For example, in Figure 5.7, the concept f_1 is built from concepts $f_{11}, f_{12}, \cdots, f_{1n}$.

Figure 5.7 A part of A O taxonomy



If the concepts f_1 and f_2 are frequent, then finding a frequent itemset consisting of f_2 and a subset of $\{f_{11}, f_{12}, \cdots, f_{1n}\}$, where cardinality > 1 is not required because the user is not interested in this grouping. He is interested in the grouping of $\{f_{11}, f_{12}, \cdots, f_{1n}\}$ as specified by the concept node f_1, which may be an AND node or an OR node. So, finding a frequent itemset consisting of f_2 and f_1 is more interesting as per the user's view point.

If the concept f_1 is not frequent, then finding frequent itemsets consisting of f_2 and any subset of $\{f_{11}, f_{12}, \ldots, f_{1n}\}$, say $\{f_{1i}, f_{1j}, f_{1k}\}$ (where cardinality > 1) is not meaningful, since there is no interest in the subset $\{f_{1i}, f_{1j}, f_{1k}\}$ as per the knowledge in the taxonomy. But in the A O taxonomy, if there exists some other frequent concept say $f_{\hat{1}}$, built from a subset of cardinality m from the set $\{f_{11}, f_{12}, \ldots, f_{1n}\}$ where $m \leq n$, then finding a frequent itemset consisting of f_2 and $f_{\hat{1}}$ could be interesting. Therefore, in the absence of $f_{\hat{1}}$, finding frequent itemsets consisting of some arbitrary grouping of f_{1i}s and f_2 is not meaningful. Hence, it is sufficent to find the frequent itemsets made up of two items/concepts (i.e., 2-nodeset) w.r.t the complete A O taxonomy and it is not necessary to find all combinations of items and concepts as in the conventional association rule mining methods. From now onwards by an A O taxonomy, we mean a complete A O taxonomy.

Next, we give an estimate for the maximum number of candidate 2-nodesets generated using this approach.

Estimation for the maximum number of candidate 2-nodesets in an A O taxonomy :

Let us assume that the A O taxonomy is a balanced tree. Let T be the height of the tree and, let r be the fanout of any node in the tree. The above assumption, (i.e A O taxonomy is a balanced tree) is justified because, we can select r and T so that the number of nodes generated in our taxonomy is always less than or equal to the number in a balanced tree for the given r and T. Further, in real life situations the values of r and T are reasonably small (say ≤ 10).

Let $n = T$ - 1.

Number of nodes under any node (including that node) at level n is given by

$1 + r + r^2 + r^3 + \ldots + r^{n-1}$

$= \frac{r^n - 1}{r - 1}$.

We know that at level n there are r siblings. So, the number of candidate 2-nodesets generated at level n is given by

$\left(\frac{r^n-1}{r-1}\right)^2 \times (r-1) + \left(\frac{r^n-1}{r-1}\right)^2 \times (r-2) + \ldots + \left(\frac{r^n-1}{r-1}\right)^2 \times 1$

$= \frac{(r^n-1)^2 \times r}{2\times(r-1)}$.

Number of candidate 2-nodesets generated at level $n-1$ is given by

$\frac{\left(r^{n-1}-1\right)^2 \times r^2}{2\times(r-1)}$.

Note that, at level $(n-1)$, under each node, the number of candidate 2-nodesets generated is

$\frac{\left(r^{n-1}-1\right)^2 \times r}{2\times(r-1)}$.

This is because, as in the level n case, there are r siblings under each node at level $(n-1)$ also. So, the total number of candidate 2-nodesets generated at level $n-1$ is

$\frac{\left(r^{n-1}-1\right)^2 \times r}{2\times(r-1)} \times r$.

Similarly at level 1, the number of candidate 2-nodesets generated is

$\frac{\left(r^1-1\right)^2 \times r^n}{2\times(r-1)}$.

Or, in general, the total number of candidate 2-nodesets generated from level n down to level 1 is given by

$\sum_{i=1}^{n} \frac{r^i\left(r^{n+1-i}-1\right)^2}{2\times(r-1)}$.

$= \bigcirc(r^{2n})$.

$= \bigcirc(r^{2T})$, since $n = T$ - 1.

Complexity and Characteristic of A O Single Scan Algorithm :

Once we find the description for each node in the taxonomy in terms of collection of items at level 1, finding frequent 2-nodesets between any nodes is sufficient because of lemma 1. If we consider the taxonomy as a balanced tree, and let **r** be the fanout of each node and **T** be the height of the tree (**n** = **T** - 1), then the number of *candidate 2-nodesets* generated is of the order of $\bigcirc(\mathbf{r}^{2n})$ in comparison with $\bigcirc(\mathbf{2}^{r^n})$ number of *all candidate itemsets* generated by any other algorithm like *Apriori* [Agrawal et.al, 94]. Because of this reduction in the number

of candidate sets, we can find the support for all 2-nodesets together in one scan. In otherwords, we need to scan the database **only once**. So we call this algorithm **A O Single Scan algorithm**.

From the above description, it is clear that number of frequent 2-nodesets generated is also small (because the number of candidate nodesets is less). This implies that the number of association rules generated is small. We can claim that association rules generated using A O Single Scan algorithm are *more interesting* because we can find the associations upto a level specified by the user, through the A O taxonomy.

Now we describe the A O Single Scan algorithm.

A O Single Scan

Input

1. Transaction database, $\mathfrak{D}$.
2. User defined minimum support, σ, and minimum confidence c.
3. A O-taxonomy, $\mathfrak{T}$ with height T and Level of interestingness, $\mathfrak{M}$ with respect to $\mathfrak{T}$. $\mathfrak{M}$ gives the upper bound on the level till which we have to mine using the A O taxonomy. By changing the value of $\mathfrak{M}$ we can change the level of interestingness. So, because of the parameter $\mathfrak{M}$, we have **interactive mining**. [Note : $1 \leq \mathfrak{M} \leq T$].

Output

Interesting association rules.

Steps

1. Find the *descriptions* of all nodes in $\mathfrak{T}$ upto the level $\mathfrak{M}$ using **Description Generation Algorithm (DGA)**. Put the *descriptions* in to the list *Q.description* and corresponding node names in to the name list, *Q.name*. The name list is used to identify the node names for a given discription.

2. Generation of candidate 2-nodesets :

$k = 1$

For $i = 1$ to $|Q|$ - 1 {

For $j = i + 1$ to $|Q|$ {

$\mathfrak{l}_i = Q_i.description$; $\mathfrak{l}_j = Q_j.description$;

If (($\mathfrak{l}_i \not\subset \mathfrak{l}_j$) and ($\mathfrak{l}_j \not\subset \mathfrak{l}_i$)) {

$\mathbb{L}_k.description = \mathfrak{l}_i \cup \mathfrak{l}_j$;

$\mathbb{L}_k.name = Q_i.name \cup Q_j.name$

$k = k + 1$

}

}

}

Explanation : For any two items/concepts in the list $Q.name$ say, $\mathfrak{L}_i$ and $\mathfrak{L}_j$, if their *descriptions* in the list $Q.description$ (say $\mathfrak{l}_i$ and $\mathfrak{l}_j$ respectively) are not subset of each other, then put $\mathfrak{l}_i \cup \mathfrak{l}_j$ in the list $\mathbb{L}.description$ and put $\mathfrak{L}_i \cup \mathfrak{L}_j$ in the list $\mathbb{L}.name$.

3. Generation of frequent 2-nodesets :

For each transaction $t \in \mathfrak{D}$ {

For $i = 1$ to $|Q|$

Compute support of $Q_i.description$ // i.e., Count(Q_i)

If Count(Q_i) $< \sigma$ then remove it from Q.

For $i = 1$ to k

Compute support of $\mathbb{L}_i.description$ // i.e., Count($\mathbb{L}_i$)

}

$i = 1$

While ($i \leq k$) {

If (Count($\mathbb{L}_i$) $\geq \sigma$)

$\mathbb{L}_i$ is frequent 2-nodeset;

$i = i + 1$

Else

Swap($\mathbb{L}_i$,$\mathbb{L}_k$); // i.e., swap the contents of $\mathbb{L}_i$ and $\mathbb{L}_k$

$k = k$ - 1

}

Note : There are k frequent 2-nodesets in $\mathbb{L}$.

Explanation : Compute the support values for the elements in the lists $Q.description$ and $\mathbb{L}.description$ using transactions in $\mathfrak{D}$. Remove names from the lists $Q.name$ and $\mathbb{L}.name$; remove descriptions from the lists $Q.description$ and $\mathbb{L}.description$ whose descriptions have support value lesser than σ.

4. Generation of minimum number of association rules :

For each 2-nodeset in $\mathbb{L}.name$ of the form $A \cup B$

If (Support($\mathbb{L}.description(A \cup B)$)/Support($Q.description(A)$) $\geq \sigma$

Output the association rule, $A \Longrightarrow B$.

(Where, $\mathbb{L}.description(A \cup B)$ is the discription of $A \cup B$ and $Q.description$(A) is the description of A).

If (Support($\mathbb{L}.description(A \cup B)$)/Support($Q.description(B)$) $\geq \sigma$

Output the association rule, $B \Longrightarrow A$.

($Q.description$(B) is the description of B).

5.5 Experimental Results

For the experimentation, we construct random A O taxonomies. We then apply **A O Single Scan** algorithm to obtain the number of frequent 2-nodesets generated and number of comparison operations performed for different randomly generated A O taxonomies.

Table 5.1 shows the parameters used in the generation of the A O taxonomies.

Table 5.1 A O taxonomy parameter values

Taxonomy set	Taxonomy height (T)	Max. Fanout/node (r)
A O taxonomy-3_6	3	6
A O taxonomy-3_8	3	8
A O taxonomy-4_6	4	6
A O taxonomy-3_10	3	10

Generation of a Random A O taxonomy :

For each taxonomy set, we fix the maximum height (T) and the maximum number of children per node (r). First, we generate the root node at level T. For each node generated, the number of children ($\leq r$), functionality (AND / OR), and the τ value ($\leq$ number of child nodes generated) if the functionality is OR, are generated randomly. We used the subtractive pseudo random number generator described in [Knuth 81] to generate the random numbers. The process of successive generation of the child nodes is stopped if either the node under consideration is at level 1 or no more children nodes are generated by the random number generator. For the nodes at level 1, we pick the items randomly from the item set (i.e., item# 0 to item# 999). The transaction database is based on the simulation of buying patterns of customers in a retail environment. We generate the synthetic data sets as described in [Agrawal et.al, 96]. Table 5.2 shows the parameters used and their settings in the generation of the synthetic data. Table 5.3 shows the number of frequent 2-nodesets

Table 5.2 Parameters

Parameters	Descriptions	Setting value
$\|D\|$	Number of transactions	100K
$\|T\|$	Average size of transactions	10
$\|I\|$	Average size of maximal potentially frequent itemsets	4
$\|L\|$	Number of maximal potential frequent itemsets	2000
NOI	Number of items	1000
σ	User specified minimum support	0.25%

and the number of comparison operations (NO_OF_COMP) performed.

For the same parameter settings as in Table 5.2, an efficient implementation of the *Apriori* algorithm [Agrawal et.al, 96] based on partitioning method [Savasere et.al,

95] generates 8453 number of frequent itemsets after phase II of the algorithm and the total number of comparison operations is $1,874,999$. The number of frequent 2-nodesets generated based on our apporach is reduced by a factor of thousand in this example when compared with the results of partition based algorithm.

Table 5.3 No. of frequent 2-nodesets and NO_OF_COMP values for different A O taxonomies

Taxonomy set	No. of frequent 2-nodesets	NO_OF_COMP
A O taxonomy-3-6	25	3285
A O taxonomy-3-8	52	5317
A O taxonomy-4-6	31	4468
A O taxonomy-3-10	66	30792

5.6 Summary

The important differences between our approach and the conventional approaches are summarized below :

1. This chapter deals with the use of knowledge for mining association rules. In the literature, knowledge in the form of 'is_a' relation is used. Here, we permit usage of other types of relations also; for example we use 'a_part_of' relation between concepts/functions.

2. We propose the use of an A O taxonomy for mining generalized association rules for the first time.

3. The proposed A O taxonomy is based on items, concepts/functions. Usage of concepts/functions helps in **relevant** and **interactive** mining of association rules. It is **relevant** because our scheme uses knowledge for mining only those concepts that are of interest to the user. It is **interactive** because user can use the controllable parameter called *level of interestingness*, $\mathfrak{M}$, which gives the height up to which one has to mine. In the conventional approaches for finding generalized association rules, there is no explicit control on the extent of generalization.

4. The knowledge structure proposed by us helps in generating relevant negative association rules in a natural and efficient manner. This is a step ahead of the existing work on mining negative associations.

5. We proposed an algorithm called **AO Single Scan**, which can be used to generate all the interesting generalized association rules between seemingly unrelated concepts and hence the obtained associations are **interesting**. This algorithm needs to scan the database **only once**. The existing algorithms at best need **two database scans.**

The important conclusions that can be drawn from our work are :

- The time complexity of A O Single Scan algorithm is of $\bigcirc(\mathbf{r}^{2n})$ against the time complexity of $\bigcirc(2^{r^n})$ associated with the conventional methods. Our Experimental results support the above claim. The number of frequent 2-nodesets generated is reduced by a factor of thousand when compared with the number of frequent itemsets generated by the conventional methods. This observation supports our claim that the A O single scan algorithm generates all the "meaningful (knowledge based)" association rules using lesser computational resources.

- **Weighted A O taxonomy** : In this chapter, we assumed that each child node of a parent node (AND/OR node) has the same weight. But due to a business strategy, more emphasis could be given to some items and less emphasis to other items. The association rules generated using this criteria are called weighted association rules [Cai et.al, 98]. Such rules can be obtained by assigning normalized weights to the child nodes. Similarly, we can have a real number as the value of τ in the case of OR node. Such an A O taxonomy is called **Weighted A O taxonomy**. With minor modifications to the A O taxonomy based algorithms this case can be easily accommodated. A simple neural network can be designed to handle such a weighted generalization scheme [Kim et.al, 00].

Chapter 6

Association Generation Using Pattern Count Tree

6.1 Introduction

In this chapter, we introduce a novel abstraction of relevant part of a database and use it for association mining. Here, we propose a collection of schemes which use either explicit or implicit knowledge along with a database as input [cf. 3.4.1] for finding associations. We also show how some of the proposed schemes can be extended to discover associations from multiple databases.

Recently, a method that employs a data structure called *frequent pattern tree* (*FP-tree*), was proposed and used to generate frequent (large) itemsets [Han et.al, 00] from a transaction database. The FP-tree is an extended prefix-tree structure storing crucial, quantitative information about frequent itemsets. In this method, two scans of database are required to generate all frequent itemsets. During the first scan, frequent 1-itemsets are generated. During the second scan, frequent-pattern tree (FP-tree) is constructed.

It is shown that the FP-tree contains the complete information of database in relevance to frequent pattern mining [Han et.al, 00]. A major problem associated with the FP-tree is that it needs two database scans. In this chapter, we propose a new tree structure which can be built using a single database scan for mining frequent itemsets. *We show*

that using the proposed tree structure, FP-tree can be constructed with less time when compared with direct construction of FP-tree from the database.

Typically, for mining association rules from large databases, it is assumed that the entire database is available. So, mining algorithms [Agrawal et.al, 93B; Han et.al, 00] are designed to work on a static transaction database. However, in real-life, transaction databases are dynamic. For example, in a supermarket database new transactions get added to the database. Incremental mining algorithms are proposed in [Cheung et.al, 96A; Cheung et.al, 97; Sarda et.al, 98; Thomas et.al, 00]. In a more general sense, the requirement for incremental mining arises when there is a change in the state of the database. This change of state is affected due to either addition/deletion of transactions to/from the existing database. The motivation for deleting a transaction comes out of the following scenario. In a supermarket, if a customer buys items, then a transaction is added to the database. However, if at a later point, if s/he is not satisfied with one or more items, s/he may return the items. In such a case, the need for deleting items from earlier transactions arises. This may be handled by deleting one or more earlier transactions and adding new transactions which suitably reflect the deletion of returned items.

Other requirements for mining in a dynamic environment is to handle change of knowledge and change of support value. Handling of change of knowledge is required because knowledge may also change as new knowledge is acquired; for example, this happens in Dewey-decimal classification system [George 91]. Similarly, in most of the mining algorithms for finding frequent itemsets, the support value is fixed and is chosen arbitrarily. This is because user has no way to find the appropriate support value for finding frequent itemsets. So, there is a need for changing support value for mining frequent itemsets. We show that the proposed tree structure handles *change of data*, *change of knowledge* and *change of support value* without revisiting the database. We call such a scenario, **dynamic mining**.

Our contributions reported in this chapter are :

Proposal of a new tree structure called Pattern Count tree (PC-tree) and its use in

generating frequent itemsets.

- *Discovering generalized frequent itemsets with the help of the generalized PC-tree using* ***a single database scan.*** *Extension of the FP-tree structure to mine generalized association rules using* ***two database scans.***

- *Generation of FP-tree from given PC-tree using less time than that of the direct construction of FP-tree from the database.*

- *Generation of modified versions of PC-tree and FP-tree based on frequent 2-itemsets because these structures can be exploited to reduce the exploration of candidate itemsets.*

- *To show that the PC-tree is a member of a class of appropriate representative structures which is both necessary and sufficient for* ***dynamic mining.*** *In this approach both addition and deletion of transactions, change of knowledge, change of support value - are considered to capture the dynamic nature of the real-life scenario.*

- *An efficient* ***distributed*** *association rule mining algorithm based on PC-trees.*

- *PC-trees based scheme for mining* ***multiple databases.***

- *PC-tree is used for other data mining applications like classification and clustering.*

The remaining part of the chapter is organized as follows. Section 6.2 introduces a novel tree structure called PC-tree and its variants. Section 6.3 introduces dynamic mining of frequent itemsets. Section 6.4 introduces distributed mining of frequent itemsets. We discuss the scheme for mining multiple databases using PC-trees in section 6.5. Here, we use a collection of databases and a semantic network as input. Section 6.6 presents our experimental study based on PC-trees. We show the other data mining applications of PC-tree like clustering and classification in section 6.7. Section 6.8 summarizes the study with conclusions.

Problem Definition

In this work, first we deal with generation of frequent itemsets from a transaction database. Here, we assume that the relation of reference already exists and further, it is a transaction database, $\mathfrak{D}$. Here, valuetable is the projection of transaction database on attribute *items_purchased* and a 'tuple' is an *itemset*. We propose a tree structure for representing the transactions of a transaction database in a compact manner. We show the adequacy of such a tree structure - for dynamic mining of frequent itemsets based on *change of data*, *knowledge* and *support value*; and distributed mining of frequent itemsets. We used the same tree structure for mining generalized frequent itemsets [Srikant et.al, 95]. We also show the discovery of association rules corresponding to the values of attributes which belong to the relations of one or more databases in set $\mathbb{D}$.

6.2 Tree Structures for Storing Patterns

Let $I = \{i_1, i_2, \ldots, i_n\}$ be a set of items. Then a transaction may be viewed as a non-empty ordered subset of I [Agrawal et.al, 93B].

Definition 6.2.1 Pattern : A *pattern* is a non-empty ordered subset of a transaction.

We view both transactions and patterns as ordered subsets, where item numbers appear in an increasing order. The relation between 'pattern' and 'transaction' is characterized by lemma 6.2.1.

Lemma 6.2.1 Every transaction of a transaction database, DB, is a pattern but every pattern need not be a transaction.

Proof

1. Part I : Let t be a transaction in DB. So, $t \subseteq I$. Consider the set of items, p $(= t)$. From the definition of pattern, p is a pattern as $p \subseteq t$.

2. Part II : Consider a transaction t in DB, where some t_1 ($\subset t$) is not a transaction in DB. Let $p = t_1$. So, p is a pattern but not a transaction. So every pattern is not a transaction.

For example, t_i : $\{19, 40, 125, 179\}$ is a transaction and one of the patterns is $\{19, 179\}$.

In the rest of the section, we define the proposed tree structure which we call the Pattern Count tree. We also deal with *variants* of Pattern count tree and FP-tree.

6.2.1 Pattern Count tree (PC-tree)

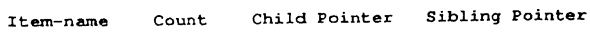
Pattern Count tree, PC-tree for short, is a data structure which is used to store all the patterns occurring in the tuples of a transaction database, where a count field is associated with each item in every pattern. In the PC-tree, all the patterns in the transaction database are stored. The count field is responsible for a compact realization of the database.

The structure, construction, properties and comparison of PC-tree with FP-tree are described below.

6.2.1.1 Structure of a PC-tree

Each node of the tree consists of the following structure: item-name, count and two pointers called child (c) and sibling (s). Figure 6.1 shows the PC-tree node structure.

Figure 6.1 PC-tree node structure



In the node, the item-name field specifies which item the node represents, the count field specifies the number of transactions represented by a portion of the path reaching this node, the c-pointer field represents the pointer to the following pattern; and the s-pointer field points to the node which indicates the subsequent other patterns from the node under consideration.

Definition 6.2.2 Child : Child node of a node with item-name A_{ij}, in a transaction t_i, is a node with item-name A_{ij+1}.

Definition 6.2.3 Sibling : Sibling of a node with item-name A_{ik}, corresponding to a transaction t_i, is A_{lk} in a transaction t_l, $(i \neq l)$, such that $A_{ip} = A_{lp}$, for $p = 1, \ldots, k-1$. In other words, we have a single subpath of length $k-1$ from root T to A_{ik-1}, corresponding to t_i and t_l. The above definition of sibling is prompted by the binary tree representation of a tree.

For example, let t_i be : $19, 40, 179$ and t_l be : $19, 40, 125$. Then nodes containing item-names 179 and 125 are siblings. The corresponding subpath is '$19, 40$'.

6.2.1.2 Construction of a PC-tree

1. Construct the root of PC-tree, T.
2. For each transaction, t in the transaction database, DB :

Begin

$prnt = T$

For each item i in t :

Begin

If any one of the child nodes say, $chld$ of $prnt$ has item-name = item number of i

- Increment the count field value of $chld$.
- Set $prnt = chld$.

Else

- Create a new node, nw.
- Set item-name field of nw to item-number of i.
- Set Count field of nw to 1.
- Attach nw as one of the child nodes of $prnt$.
- Set $prnt = nw$.

End.

End.

We give below an example to explain the construction of a PC-tree.

Consider the transaction database shown in Table 6.1.

Table 6.1 Transaction database, D

Transaction ID	Item numbers of items purchased (transactions)
t_1	19, 40, 125, 179, 510, 527, 790, 795, 799
t_2	19, 40, 179
t_3	19, 40, 125, 510, 520
t_4	527, 740
t_5	527, 740, 795
t_6	510, 527, 790, 795

We use the equivalent binary tree representation [Horowitz et.al, 83] to represent the PC-tree and its variants. The PC-tree for the transaction database, D, is shown in Figure 6.2A.

Figure 6.2 Tree structures for storing patterns

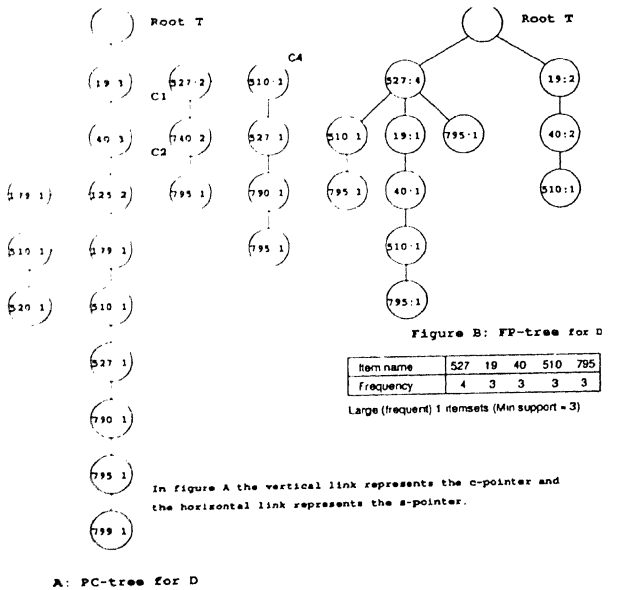


Figure 6.2B shows the FP-tree [Han et.al, 00], for the same transaction database D

shown in Table 6.1 with support value = 3. [The header table and the link between the same nodes in the tree are not shown for simplicity].

6.2.1.3 Properties of a PC-tree

We show the completeness of a PC-tree with respect to the transaction database using lemma 6.2.2.

Definition 6.2.4 Complete representation A representation R is *complete* with respect to a transaction database DB, if every pattern in each transaction in DB, is present in R.

Lemma 6.2.2 PC-tree is a complete representation of the database.

Proof Follows from construction of the PC-tree. Every transaction in the database is present in one path from root, T, to a leaf node in the PC-tree.

So, PC-tree is a *complete* representation of the database.

Definition 6.2.5 Compact representation Let D be a database of size D_s. Then, a complete representation R of DB is *compact* if $R_s < D_s$, where R_s is the size of R.

Lemma 6.2.3 PC-tree is a compact representation of the database.

Proof There are three factors responsible for the reduction in the size of the PC-tree compared to that of D.

1. From each tuple in D, we ignore some entries like TID (transaction-id) and CID (customer-id) in constructing the PC-tree. More specifically, the pattern in each tuple of DB corresponding to the attributes of interest is considered to construct the PC-tree.

2. We use only item numbers and not item names in constructing the PC-tree. Item numbers requires less space than item names.

3. If two transactions have the same prefix, then they share the same sub path in the PC-tree.

So, *PC-tree is a complete and a compact representation of the database.*

Most of the incremental algorithms are order-dependent. However, the incremental PC-tree construction algorithm is independent of order of transactions. Lemma 6.2.4 deals with this property.

Lemma 6.2.4 PC-tree is independent of order of the transactions in the transaction database, DB.

Proof We have to show that the PC-tree structures for different orders of the transactions in the database are equivalent.

Let there be n transactions in the transaction database DB. So there are $n!$ orders of presenting the n transactions. Let O_i and O_j be two different orders of the n transactions and let PC_i and PC_j be the corresponding PC-trees. Even though O_i and O_j are different ordered sets of transactions, they represent the same set of patterns. Because PC-tree is a complete representation, every pattern in PC_i is present in PC_j and every pattern in PC_j is present in PC_i. So, PC_i and PC_j are equivalent representations.

We use the result stated in lemma 6.2.4 to support the adequacy of the PC-tree for dynamic mining.

6.2.1.4 Comparison of PC-tree and FP-tree

PC-tree and FP-tree share the following properties.

1. Let $A_1, A_2, \ldots, A_k$ be the item-names of nodes in a path of a PC/FP-tree. Let $C_1, C_2, \ldots, C_k$ be the values in the count fields of above nodes respectively. Then, $C_1 \geq C_2 \cdots \geq C_k$.

2. Let the ordered set $\mathfrak{A} = \{A_{i1}, A_{i2}, \ldots, A_{il}\}$ correspond to a path in the PC/FP-tree from the root till some leaf node A_{il}. Then, any subset of $\mathfrak{A}$ of size k ($\leq l$), is a k-itemset.

For example, corresponding to the path $\{510, 527, 790, 795\}$ in the PC-tree shown in Figure 6.2A, possible 3-itemsets are $\{510, 527, 790\}$, $\{510, 527, 795\}$, $\{510, 790, 795\}$, $\{527, 790, 795\}$.

The differences between PC-tree and FP-tree are :

1. A pattern in FP-tree is based on frequent 1-itemsets and ordering of items in the transaction is unimportant. The items in the patterns are ordered in the support-descending order. More frequently occurring items are arranged closer to the root of the FP-tree. On the other hand, the PC-tree represents all itemsets. Since the items in a transaction are ordered in the ascending (or descending) order of the item number, the same order also appears in the PC-tree.

2. Typically, the number of nodes in an FP-tree is smaller than the number of nodes in the corresponding PC-tree. For example, in Figure 6.2, the PC-tree has 20 nodes and the corresponding FP-tree has only 12 nodes. However, our experimental results (**Experiment 1**, in **Section 6.6**) show that as the database size increases, there is *no significant difference* between the sizes of PC-tree and FP-tree structures.

3. The potential advantage of the PC-tree is that it can be used for *dynamic mining*, since it is a complete representation of the database. However, the *FP-tree* is *not* a *complete* representation of the database as it is based on frequent 1-itemsets. As a consequence, it is *not suited* for *dynamic mining*. We discuss dynamic mining of frequent itemsets in section 6.3.

4. Total number of disk block accesses for generating frequent itemsets using PC-tree and FP-tree are formulated as follows :
   Let B_{DB} be the number of disk blocks required to store the database, DB. Let FP and PC be the FP-tree and PC-tree built using DB respectively. Let B_{FP} be the number of disk blocks required to store FP. Let B_{PC} be the number of disk blocks required to store PC. Let B_{MM} be the size of main memory available to store any tree-structure, in terms of disk blocks.

Number of disk block accesses required to construct frequent itemsets using FP-tree is $2 \times B_{DB} + \rho_{FP} \times 2$, where ρ_{FP} is the number of disk blocks corresponding to the part of FP, which does not fit into the main memory. The multiplicative factor 2 associated with ρ_{FP} is to capture both reading and writing operations. Two database scans to construct a FP-tree is indicated by the multiplicative factor 2 associated with B_{DB}.

$$\rho_{FP} = \begin{cases} B_{FP} - B_{MM} & \text{if } B_{FP} > B_{MM} \\ 0 & \text{otherwise} \end{cases}$$

Number of disk block accesses required to construct frequent itemsets using PC-tree is $B_{DB} + \rho_{PC} \times 2$, where ρ_{PC} be the number of disk blocks corresponding to the part of PC, which does not fit into the main memory. The multiplicative factor 2 associated with ρ_{PC} is to capture both reading and writing operations.

$$\rho_{PC} = \begin{cases} B_{PC} - B_{MM} & \text{if } B_{PC} > B_{MM} \\ 0 & \text{otherwise} \end{cases}$$

Number of disk block accesses required to generate frequent itemsets using PC-tree is always less than that of FP-tree provided $B_{DB} + 2 \times \rho_{FP} > 2 \times \rho_{PC}$.

6.2.2 Construction of the LOFP-tree from a Given PC-tree

The PC-tree corresponding to a given database is unique, where as the FP-tree corresponding to a given database need not be unique. If all items in each transaction of the database are ordered lexicographically, and frequent 1-itemsets are also ordered lexicographically, when the items are having same frequency, we call the corresponding FP-tree the Lexicographically Ordered FP-tree (LOFP-tree) which is unique. Since PC-tree is a complete representation of the database, LOFP-tree can be constructed directly from the PC-tree. The construction process is carried out as follows :

1. Construct the PC-tree (as outlined in section 6.2.1.2) using one database scan.

Simultaneously, compute frequent 1-itemsets [Agrawal et.al, 93B].

2. Select a path *pth* from the PC-tree, generate all transactions out of *pth* and from each transaction retain only the frequent items using frequent 1-itemsets; use them to generate the relevant part of the LOFP-tree. This step is similar to that of construction of FP-tree [Han et.al, 00].

For the transaction database shown in Figure 6.3A, the frequent 1-itemsets generated by FP-tree and LOFP-tree are shown in Figure 6.3B and 6.3C. The FP-tree structure and LOFP-tree structure are shown in Figure 6.4. Items in the transaction of a transaction database (Figure 6.3A) are ordered lexicographically for constructing the LOFP-tree.

Figure 6.3 An example transaction database and frequent 1-itemsets

Transaction Id	Items purchased (transactions)
t1	p, m, a, c, d, f, g, i
t2	m, o, a, c, b, f
t3	j, b, f, h, o
t4	p, b, c, k, s
t5	c, a, e, f, l, m, p, n
t6	b

FIGURE A : An example transaction database

Item name	f	c	b	a	m	p
Frequency	4	4	4	3	3	3

FIGURE B : Large 1-itemsets for FP-tree
(Minimum support = 3)

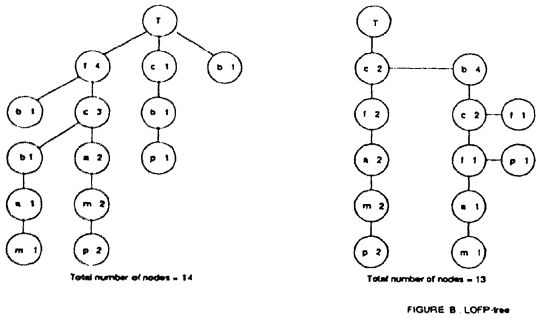
Item name	b	c	f	a	m	p
Frequency	4	4	4	3	3	3

FIGURE C : Large 1-itemsets for LOFP-tree
(Minimum support = 3)

We discuss the generation of frequent itemsets from LOFP-tree in section 6.2.5.

Our experimental results (**Experiment 2**, in section 6.6) show that LOFP-tree can be constructed *faster* using PC-tree than that of its direct construction from the database.

Figure 6.4 FP-tree and LOFP-tree



6.2.3 Generalized Pattern Count tree (GPC-tree)

Generalized association rules based on *is_a* hierarchies are introduced in [Srikant et.al, 95]. For example, $Brand\_A$ *is_a* $Soft\_drink$, and from an association rule, $Brand\_A \Longrightarrow Chips$, one can infer a generalized rule, $Soft\_drink \Longrightarrow Chips$. The use of taxonomy information for mining multiple-level association rules is exploited in [Han et.al, 95].

A taxonomy based on functionality, and a restricted way of generalization of items is discussed in chapter 5. This taxonomy (called *A O taxonomy*), provides a natural way to generalize the items, since it is based on a semantic-grouping.

In this subsection, we show how one can use the PC-tree for creating generalized itemsets, which can be used for generating generalized association rules.

The generalized PC-tree (GPC-tree) is a generalized version of PC-tree. The construction of GPC-tree is as follows :

1. Input : Transaction database, D, and an *is_a* hierarchy, which shows the grouping of items based on the domain knowledge.

2. Output : GPC-tree.

3. Steps :

   For each transaction, t in D, find the generalized transaction $\acute{t}$ using the *is_a* hierarchy.

- Use $\acute{t}$, ignore the repeation of items in $\acute{t}$ and construct PC-tree as discussed in the section 6.2.1.2.

The *is_a* hierarchy used in building the GPC-tree is shown in Figure 6.5A. Corresponding GPC-tree based on database, D, given in Table 6.1 is shown in Figure 6.5B.

Figure 6.5 Generalization process

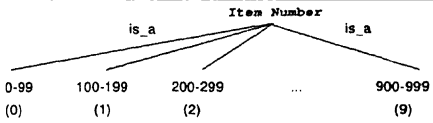


Figure A : Is_a hierarchy for item number

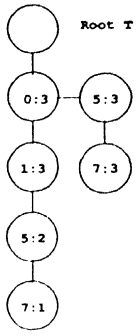
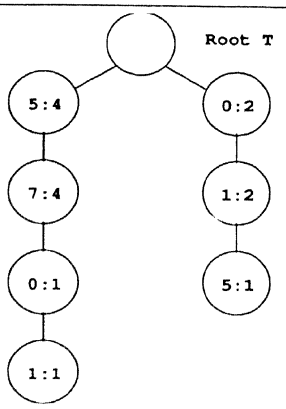


Figure B : GPC-tree for D using is_a hierarchy shown in above figure, Figure A

A meaningful question to address at this point is, "why not generalize the database and generate a generalized FP-tree from it ?" Since the proponents of FP-tree did not consider the issue of generalization, we extend the LOFP-tree to generate a generalized LOFP-tree, which we call *GLOFP-tree*. This approach requires two database scans. In the first scan, generalized frequent 1-itemsets are obtained. In the second scan, GLOFP-tree is constructed using generalized transactions. This amounts to constructing the LOFP-tree corresponding to the generalized transaction database.

Figure 6.6 shows the GLOFP-tree for the database given in Table 6.1 using the *is_a* hierarchy shown in Figure 6.5A.

Figure 6.6 GLOFP-tree for the database in Table 6.1 and is_a hierarchy shown in Figure 6.5A



6.2.4 Modified Pattern Count tree (MPC-tree)

The LOFP-tree structure is based on frequent 1-itemsets. But, for the construction of an association rule we need a large set having at least two items, i.e., a frequent 2-itemset. The number of nodes in the tree will be lesser, if we construct the tree based on frequent 2-itemsets. This observation motivated us to construct tree structures based on frequent 2-itemsets. We call such a modified version of PC-tree, modified PC-tree, or *MPC-tree* for short. In a similar manner the LOFP-tree structure can also be modified. We call this modified structure MLOFP-tree. It takes two database scans to generate either the MLOFP-tree or the MPC-tree. However, the sizes of MLOFP-tree and MPC-tree are smaller than those of LOFP-tree and PC-tree. In fact, MLOFP-tree is a subtree of the LOFP-tree and MPC-tree is a subtree of the PC-tree. Algorithms for generating these modified tree structures are given below.

MPC-tree generation

1. Input : Transaction database, D.
2. Output : MPC-tree.
3. Steps :

- Scan the database to generate frequent 2-itemsets.
- For each transaction in the database, D, select the items that are part of any frequent 2-itemset generated in the above step.
- Construct the PC-tree based on the items which are selected in the above step, as discussed in section 6.2.1.2.

MLOFP-tree generation

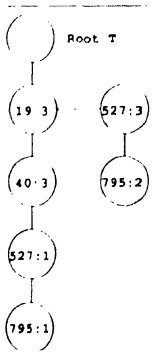
1. Input : Transaction database, D.
2. Output : MLOFP-tree.
3. Steps :
   - Scan the database to generate frequent 2-itemsets.
   - For each transaction in the database, D, select the items that are part of any frequent 2-itemset generated in the above step.
   - Construct the LOFP-tree based on the items which are selected in the above step, as discussed in section 6.2.2.

To generate frequent 2-itemsets, we used Upper Triangular Matrix (UTM) to store 2-itemsets. The UTM for the database D given in Table 6.1. is shown in Table 6.2. Note that, the diagonal entries in this table indicate frequencies of 1-itemsets. For example, for item number 19, in the first row and the first column, the entry is 3 which means that the frequency of 19 is 3. For the transaction database shown in Table 6.1, with the user defined support value = 3, and frequent 2-itemsets obtained from UTM, Figure 6.7 shows the MPC-tree. Note that number of nodes in the corresponding FP-tree (see Figure 6.2B) is 12 where as the number of nodes in MPC-tree is only 7.

Table 6.2 UTM for 2-itemsets

	19	40	125	179	510	520	527	740	790	795	799
19	3	3	2	2	2	1	1	0	1	1	1
40		3	2	2	2	1	1	0	1	1	1
125			2	1	2	1	1	0	1	1	1
179				2	1	0	1	0	1	1	1
510					3	1	2	0	2	2	1
520						1	0	0	0	0	0
527							4	2	2	3	1
740								2	0	1	0
790									2	2	1
795										3	1
799											1

Figure 6.7 MPC-tree for the database in Table 6.1 and user defined minimum support value = 3



6.2.5 Frequent Itemset Generation

For generating frequent itemsets, construct the frequent 1-itemset list (L) in the descending order of their frequencies in parallel with construction of PC-tree. For each entry a_i in L, link all the nodes of the PC-tree (through node-links) which are having a_i as their item-name. To find all possible frequent patterns that contain a_i, follow a_i's node-links and obtain the patterns where a_i is the last element in it. If the count value of a_i in all patterns together is greater than or equal to the user defined minimum support value (σ), then pick the sub pattern (which is the pattern excluding a_i) from each pattern with the count value set to the count value of a_i in that pattern. Find the intersection of all sub patterns and add the count value of corresponding item in the sub pattern. If each item in the resultant sub pattern has its count value greater than or equal to σ, then the resultant pattern is a frequent x-itemset, where x is the number of items in the sub pattern.

Pragmatically, we construct the LOFP-tree using PC-tree and large 1-itemsets; LOFP-tree can be used to generate frequent itemsets using the scheme proposed by [Han et.al, 00].

6.3 Dynamic Mining of Frequent Itemsets

In this section, we describe the mining procedure for generating frequent itemsets which characterize *change of data*, *change of knowledge* and *change of support value*. In the literature on association rule mining, mining under change of data scenario is called *incremental mining*. We explain how PC-tree handles incremental mining in section 6.3.1, change of knowledge in section 6.3.2 and change of support value in section 6.3.3.

6.3.1 Change of Data (Incremental Mining)

Incremental mining is a method for generating frequent itemsets over a changing transaction database in an incremental fashion without considering the part of the database which is already mined. This is a non-trivial method because a database may allow

frequent or occasional updates like addition or deletion of transactions. These updates may not only invalidate some existing frequent itemsets, but also turn some non-frequent itemsets to be frequent.

In real-life, the databases built for supermarkets, reservation systems, etc., are dynamic in nature. That means, there is a continuous updation of the existing database. The updation of the supermarket transaction database may be due to the purchase of set of items or returning some of the purchased items (if the customer is not satisfied with the items). Similarly, in the transaction database for a reservation system, the updation is not only due to the purchase of tickets but also due to cancellation of tickets. In this section, we show that PC-tree discussed in subsection 6.2.1.2 is suitable for incremental mining of frequent itemsets which handles both *addition* and *deletion* of transactions.

There are three operations which lead to changes in a database.

1. **ADD** :- A new transaction is added to the transaction database : this is handled in a PC-tree either by incrementing the count value of the nodes corresponding to the items in the transaction or by adding new nodes or by performing both the above activities.

   Algorithm for ADD : Refer to the construction of the PC-tree, discussed in subsection 6.2.1.2.

2. **DELETE** :- An existing transaction is deleted from the transaction database : this is handled in a PC-tree by decrementing the count value of the nodes corresponding to the items in the transaction, if their count value is > 1; or by deleting the existing nodes corresponding to the items in the transaction.

   Algorithm for DELETE:

   (a) Input :

   i. Transaction t_x which is to be deleted.

   ii. PC-tree with root = T.

(b) Output : Updated PC-tree.

(c) Steps :

i. Identify the list of nodes in the PC-tree corresponding to t_x. Note that the first element in t_x appears as a child of T. Call the list, *path*.

ii. Decrement the count field value of nodes along *path*.

iii. Use *path*, which is pointing to the list of nodes corresponding to t_x in PC-tree, remove all nodes in the *path*, where the count value 0 is encountered.

3. **MODIFY** :- Some of the items in an existing transaction are deleted : this situation may occur if the customer returns some, but not all, of the items purchased in a transaction. This is handled by deleting the existing transaction (which is handled as in operation 2 above) and adding new transaction (which is handled as in operation 1 above) to reflect the deleted items.

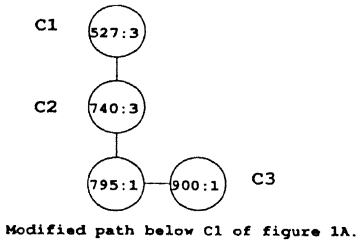
Hence an incremental mining algorithm needs to handle addition of a transaction (operation 1 above) and deletion of a transaction (operation 2 above).

Now we examine the role of PC-tree in incremental mining. In incremental mining, the addition of new transactions to the database will change the PC-tree structure appropriately without having to scan the database again. The changes required in the PC-tree are minor in nature as addition of transactions leads to either change in the count value of some of the existing nodes or addition of new nodes in the PC-tree or both. For example, consider an addition of transaction, t_7, given below to the database shown in Table 6.1:

Transaction ID	Item numbers of items purchased
t_7	527, 740, 900

Integration of t_7 into the PC-tree shown in Figure 6.2A leads to changes in the path below node $C1$. The modified path below $C1$ is shown in Figure 6.8. Observe that there is both a change in the count value of nodes $C1$ and $C2$ and addition of node $C3$ as a child node of $C2$.

Figure 6.8 A portion of PC-tree after addition



Consider the deletion of transaction t_5 in the database shown in Table 6.1 with respect to PC-tree shown in Figure 6.2. We need to nullify the effect of t_5. This is achieved by decrementing the count value of the nodes having item number 527 and 740; and deleting the node having item number 795, since its modified count value is 0.

Now consider the deletion of item 527 from the transaction t_6 in the database shown in Table 6.1. This is achieved by deleting transaction t_6 : {510, 527, 790, 795 } and adding the new transaction, t_6' :{510, 790, 795 } to the PC-tree shown in Figure 6.2. This leads to changes in the path below the node $C4$ in Figure 6.2. The modified path below $C4$ is shown in Figure 6.9. Observe that the node with item number 527 in the path below the node $C4$ of Figure 6.2 does not exist in the modified path shown in Figure 6.9.

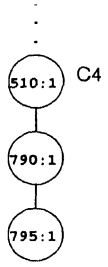
There is one more kind of updation of a transaction database. This corresponds to increase in the number of items available. For example, launching of a new product may necessitate the need to accommodate it in the supermarket. Henceforth, transactions may have this item also. This situation can be viewed as adding new transactions to the database. This addition can be reflected in the PC-tree using operation **ADD**.

Typically, incremental algorithms depend on the order in which data is presented [Elmasri et.al, 00]. However, PC-tree can be built incrementally and it is *independent of the order of transactions* as shown in lemma 6.2.4.

6.3.1.1 Representations to handle incremental mining

A transaction can be represented in different ways. For example, each transaction can be either represented as a binary vector or as a list of items. In the binary case, a 1/0 in a

Figure 6.9 A portion of PC-tree after deletion



Modified path below C4 of Figure 1A.

position corresponds to the presence/absence of the corresponding item in the transaction. These two representations are equivalent for mining frequent itemsets.

Equivalent representations : Two representations R_1 and R_2 of a transaction database DB are *equivalent*, if one can obtain R_1 from R_2 and vice versa.

Even though equivalence is a desirable property, there are several operations for which it is not needed. For example, if a database DB is mined to generate a set of frequent itemsets L, then DB and L are not equivalent. This is because we cannot construct DB from L. However, DB and L are equivalent conditionally, that is, in terms of frequent itemsets. In this work, our aim is to mine frequent itemsets. So, we are interested in *frequent-itemset equivalence.*

Definition 6.3.1.1 Frequent-itemset equivalent : Let DB be a transaction database with L as the set of all frequent itemsets in DB. Let R_1 and R_2 be any two representations of DB. Let L_1 and L_2 be the sets of frequent itemsets generated from R_1 and R_2 respectively. R_1 and R_2 are said to be *frequent-itemset equivalent* if $L_1 = L_2 = L$.

Note that PC-tree and LOFP-tree are two different representations of a given database DB and are frequent-itemset equivalent.

In dynamic updation, a database DB is updated to $D\acute{B}$, such that $D\acute{B} = DB + db_1 - db_2$, where db_1 is the set of transactions added and db_2 is the set of transactions deleted.

In incremental mining, we would like to update the frequent-itemset equivalent representation R of DB to $\acute{R}$ using only R, db_1 and db_2. Further, $\acute{R}$ should be frequent-itemset equivalent to $D\acute{B}$. We say that such a representation is appropriate for incremental mining.

Definition 6.3.1.2 Appropriate representation for incremental mining : Let R be a frequent-item equivalent representation of DB. Let DB be updated to $D\acute{B}$ such that $D\acute{B} = DB + db_1 - db_2$. R is an *appropriate representation for incremental mining*, if there exists an algorithm that generates $\acute{R}$ using only R, db_1 and db_2 such that $\acute{R}$ is frequent-itemset equivalent to $D\acute{B}$.

The above definition guarantees that an "incremental mining" algorithm does not scan the database DB more than once. However, it permits multiple scanning of db_1 and db_2. In our approach, we construct the PC-tree incrementally by scanning DB, db_1 and db_2 only once.

From the above explanation, it is clear that PC-tree is an appropriate representation for incremental mining. The reason is, PC-tree is not only a frequent-itemset equivalent representation but also has **ADD**, **DELETE** and **MODIFY** algorithms which generate updated PC-tree to represent the modified database and the updated PC-tree is frequent-itemset equivalent to the modified database. Further, PC-tree can be updated incrementally using one database scan unlike the incremental approach proposed in [Cheung et.al, 97]. On the other hand, LOFP-tree is not an appropriate representation for incremental mining. The reason is, even though it is a frequent-itemset representation of the database, it is not possible to generate a modified LOFP-tree to represent the modified database by just using the LOFP-tree and the fraction of the database (db_1 - db_2) which changes the database.

Definition 6.3.1.3 Pattern equivalent : Let DB be a transaction database with P as the set of all patterns in the transactions in DB. Let R_1 and R_2 be any two representations of DB. Let P_1 and P_2 be the sets of patterns generated from R_1 and R_2 respectively. R_1 and R_2 are said to be *pattern equivalent* if $P_1 = P_2 = P$.

A pattern equivalent representation R of a database DB is both necessary and sufficient for incremental mining of frequent itemsets. Note that PC-tree is a pattern equivalent representation of database DB, whereas LOFP-tree is not.

Lemma 6.3.1.1 Pattern equivalence is sufficient but not necessary for frequent-itemset equivalent.

Proof From the definition of the pattern equivalence, it is clear that pattern equivalent representation has all the patterns in the transactions that the database has. From this, the frequent itemset which we derive from the pattern equivalent representation is identical to the one which we derive from the database. So, pattern equivalent representation is frequent-itemset equivalent. On the other hand, if the representation is frequent-itemset equivalent, then we cannot derive all the patterns that the database has. Because a frequent-itemset equivalent representation not have the patterns which are not frequent. Hence if the representation is frequent-itemset equivalent then it need not be pattern equivalent.

Lemma 6.3.1.2 PC-tree is an appropriate representation for incremental mining.

Proof We know that PC-tree is a pattern equivalent representation. So, it is frequent-itemset equivalent from the above lemma. Moreover, we have algorithms Add, Delete and Modify - that change the PC-tree, PC, corresponding to the database DB to the updated PC-tree, PC', corresponding to the modified database DB' using only PC , db_1 and db_2, where $DB' = DB + db_1 - db_2$. So PC-tree is an appropriate representation for incremental mining from definition 6.3.1.2.

6.3.2 Change of Knowledge

Another requirement for mining in a dynamic environment is to handle change of knowledge. Knowledge representation in the mining process is mainly by an 'is_a' hierarchy. So, change of knowledge leads to deletion of existing nodes or creation of new nodes or moving children nodes of one node to some other in the same or in a different level in the 'is_a' hierarchy. A self illustrative example is shown in Figure 6.10.

Figure 6.10 Change of knowledge trees

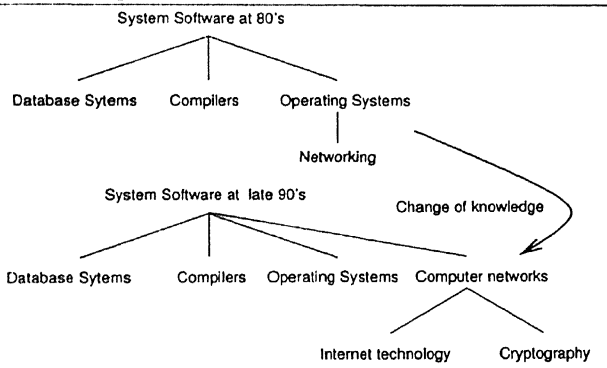


FIGURE A : Movement of node, Networking as a new branch of System software.

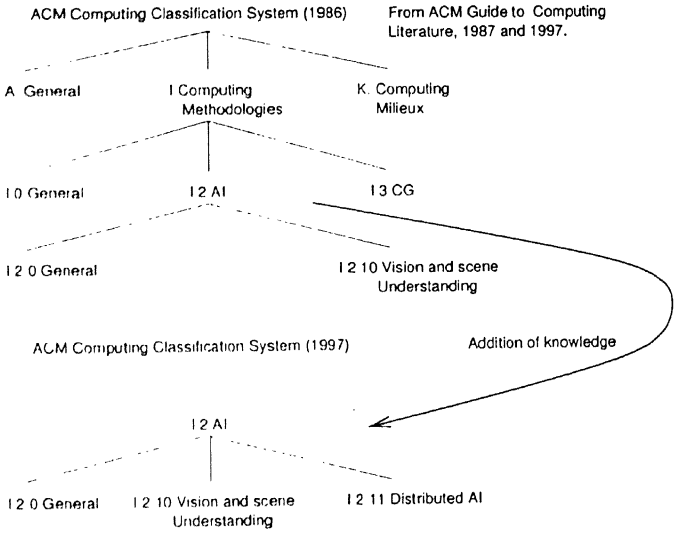


FIGURE B : Addition of new knowledge (Distributed AI), an example from ACM Computing Classification System

Let K_1 be the knowledge represented using an 'is_a' hierarchy. Let PC be the PC-tree for the database D. We can construct generalized PC-tree (GPC-tree discussed in section 6.2.3) by applying K_1 to the transactions obtained from PC without accessing D. If knowledge K_1 is changed to K_2, then by applying K_2 on PC, we get another generalized PC-tree without accessing D. This shows that PC-tree handles dynamic knowledge.

6.3.3 Change of User Defined Minimum Support Value

In most of the mining processes for finding frequent itemsets, support value is fixed and is chosen arbitrarily. This is because the user has no way to find the appropriate support value for finding frequent itemsets, since mining process itself is aimed at finding unknown patterns. But the point is, all unknown patterns are not useful and some even may be trivial. Because of this, some algorithms [Hidber, 99; Han et.al, 95] use variations in the user defined support value for mining patterns. Since PC-tree is complete w.r.t the database, it can handle this situation also.

Definition 6.3.1 A representation R is appropriate for dynamic mining of frequent itemsets if it satisfies the following properties :

1. R is a complete and compact representation of the database D.
2. R can be used to handle changes in database, knowledge and user define minimum support value without scanning D to generate frequent itemsets.

6.4 Distributed Mining

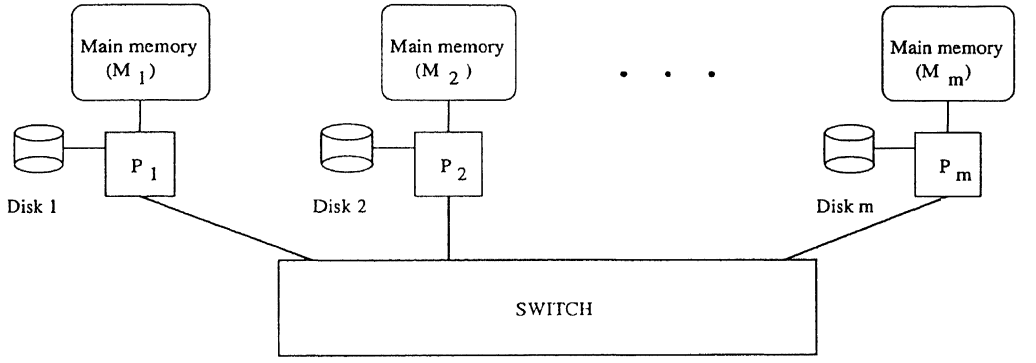
In this section we give a distributed architecture and an efficient distributed structure generation algorithm based on PC-tree where, each of the transaction database in the distributed environment is scanned only once. As a consequence, the time taken for constructing the LOFP-tree from the PC-tree, at each node in the distributed architecture is less than that for constructing LOFP-tree directly from the database. We give the

timing requirements of the distributed algorithm and its sequential counterpart along with efficiency of parallelism in the experiment section (Section 6.6).

Architecture

We propose an architecture based on Distributed Memory Machines (DMMs). Let there be m processors $P_1, P_2, \ldots, P_m$, which are connected by a high speed gigabit switch. Figure 6.11 shows this architecture. Each processor has a local memory and a local disk. The processors can communicate only by passing messages. Let

Figure 6.11 Architecture



$DB^1, DB^2, \cdots, DB^n$ be n transaction databases at n sites and be partitioned in to m non-overlapping blocks $D^1, D^2, \cdots, D^m$, where m is the number of processors available ($m \geq n$). Each block D^i has the same schema. Let t^i_j be the j^{th} transaction in partition D^i.

Algorithm : Distributed PC to LOFP tree Generator (DPC-LOFPG)

Steps :

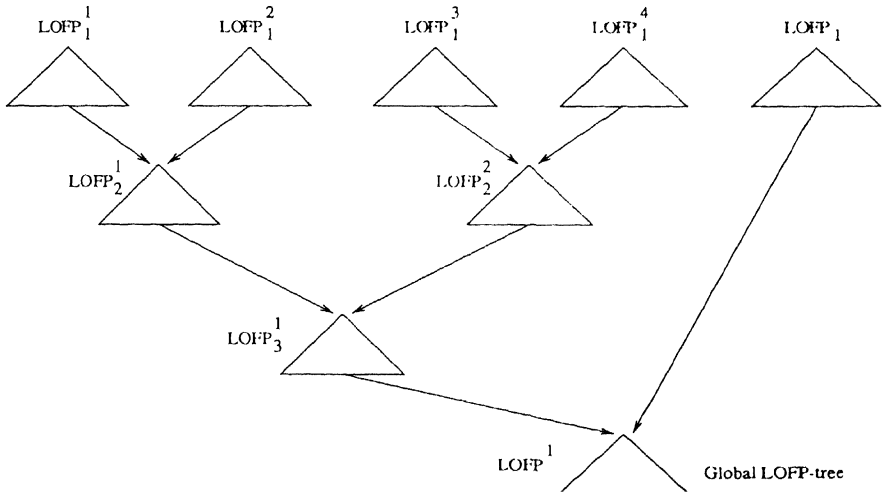
1. Each processor P^i makes a pass over its data partition, D^i and generates a local PC-tree, PC^i_1 at level 1. At the end of this process, each processor generates a *local 1-itemset vector, LV*. It is an array of size s (where s is the number of items) whose each entry LV^i[k] has a count value, corresponding to item k in the data partition D^i.

2. Process P^i exchanges local 1-itemset vector, LV^i with all other processors to develop a global 1-itemset vector, GV^i. Note that all $GV^i s(= GV)$ are identical and are frequent 1-itemsets. The processors are forced to synchronize at this step.

3. Each processor P^i constructs a local LOFP-tree using local PC_1^i and GV^i. The processors are forced to synchronize at this step.

4. Every two alternative LOFP-trees (viz. $LOFP_i^1$ and $LOFP_i^2$, $LOFP_i^3$ and $LOFP_i^4$ etc.) are given to a processor for merging and to generate a single merged LOFP-tree. This gives the LOFP-tree at the next level (viz. $i+1^{th}$ level). After every iteration the processors are forced to synchronize. During each iteration, number of LOFP-trees generated and number of processors needed are reduced by a factor of 2 w.r.t the previous iteration. Iteration stops when one (global) LOFP-tree is generated. For example, consider five LOFP-tree that are generated at Stage 3. Figure 6.12 shows the hierarchical processing activity.

Figure 6.12 An illustrative example



5. Generation of conditional LOFP-tree to generate frequent itemsets as explained in [Han et.al, 00].

6.5 Mining Multi-databases Using PC-trees

In this section, we describe the methodology based on PC-trees to discover the associations between sets of values of attributes which belong to relations of different databases.

Algorithm (MMUP)

Input :

1. Two characteristic concepts (CCs) A and B occurring in the given semantic network, $\mathcal{S}$.

2. A collection, n, of databases $D_1, D_2, \ldots, D_n$. Every one of these databases can be described by a collection of nodes in $\mathcal{S}$.

3. Generalization tree, T_A of an attribute corresponding to the concept A and a level value l. (The level value gives up to what level we have to do the generalization). Let ρ be the number of generalized values at level l.

4. User defined minimum support, σ and minimum confidence c.

Output :

Associations between values of characteristic attributes corresponding to characteristic concepts A and B.

Steps :

1. **Concept identification :**

   INPUT : A, B and the semantic network, $\mathcal{S}$

   OUTPUT : Concepts $C_1, C_2, \ldots, C_n$ linking A and B in the semantic network.

   STEPS : Refer to the module I of section 4.3.

2. **Relation (Schema) identification :**

   INPUT : Concepts $C_1, C_2, \ldots, C_n$

   OUTPUT : Relations $\mathfrak{R}_1, \mathfrak{R}_2, \ldots, \mathfrak{R}_m$ which are relevant for linking; and attribute corresponding to A i.e., CA_A is in $\mathfrak{R}_1$ and that of B i.e., CA_B is in $\mathfrak{R}_m$.

   STEPS : Refer to the module **II** of section 4.3.

3. **Trimming :**

   INPUT : $\mathfrak{R}_1, \mathfrak{R}_2, \ldots, \mathfrak{R}_m$

   OUTPUT : $\mathbb{R}_1, \mathbb{R}_2, \ldots, \mathbb{R}_m$

   STEPS : Refer to the module **III-A** of section 4.3.

4. **Creation of PC-trees**

   INPUT : $\mathbb{R}_1, \mathbb{R}_2, \ldots, \mathbb{R}_m$

   OUTPUT : PC-trees, $PC_1, PC_2, \ldots PC_m$

   STEPS :

   (a) Each $\mathbb{R}_i$ has two sets of attributes. If $(i = 1)$, then the first set corresponds to characteristic attributes and the second set corresponds to common set of attributes. If $(i = m)$, then the first set corresponds to common set of attributes and the second set corresponds to characteristic attributes. If $(1 < i < m)$, then both sets correspond to characteristic attributes.

   (b) For each relation, $\mathbb{R}_i$, where $1 < i < m$, use the values of first set and second set of attributes in that order and construct PC-trees, $PC_1, PC_2, \ldots, PC_m$ as discussed in section 6.2.1.2.

5. **Linking PC-trees**

   INPUT : $PC_1, PC_2, \ldots PC_m$

   OUTPUT : Linked PC-trees.

STEPS :

Use the values corresponding to the second set of attributes in each relation, $\mathbb{R}_i$, $(1 \leq i < m)$ to link the PC-trees.

For the relation $\mathbb{R}_i$, $i \leq i \leq m$, shown in Figure 6.13, the linking of PC-trees is shown in Figure 6.14.

Figure 6.13 Schemas of the selected relations

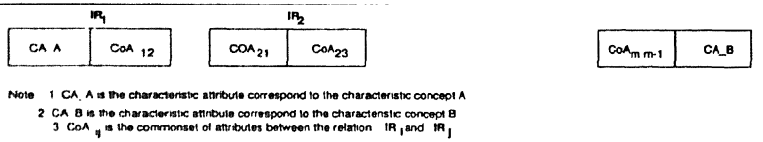
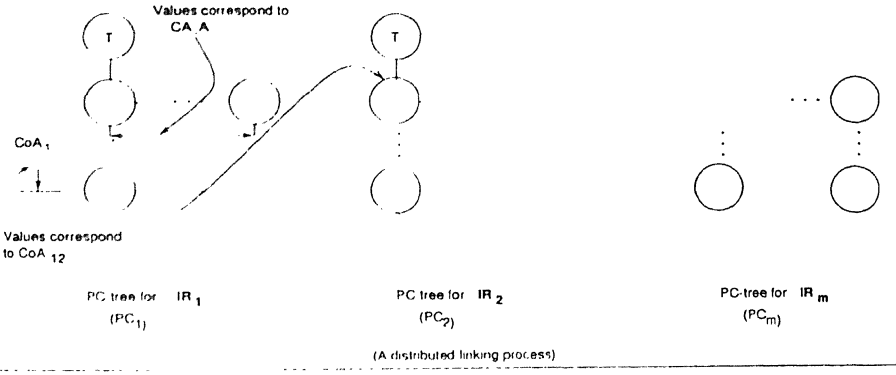


Figure 6.14 Linking of PC-trees



We explain the creation of PC-trees and their linking using following example. Consider that there are two input relations to step 4 which are obtained from previous steps. They are shown in Figure 6.15. The values corresponding to the attribute *Age* in relation $\mathbb{R}_1$ shown in Figure 6.15 are generalized using generalized tree for *Age* shown in Figure 6.15 to three values, 1, 2 and 3. We want to find associations between values correspond to attributes *Age* (i.e., *CA_A*) and *Goods_purchased* (i.e., *CA_B*).

Figure 6.16 shows the PC-trees pertaining to $\mathbb{R}_1$ and $\mathbb{R}_2$. In Figure 6.16, the dotted curved lines show the linking of PC-trees. This figure also shows the relevant *Goods_purchased* for *Age* = 1.

Figure 6.15 Relations, $\mathbb{R}_1$ and $\mathbb{R}_2$

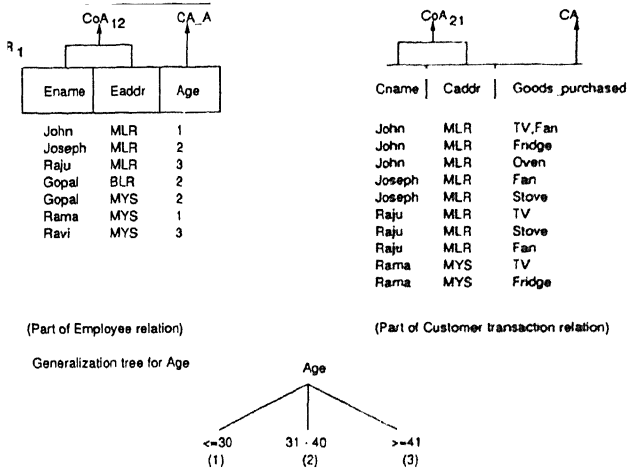
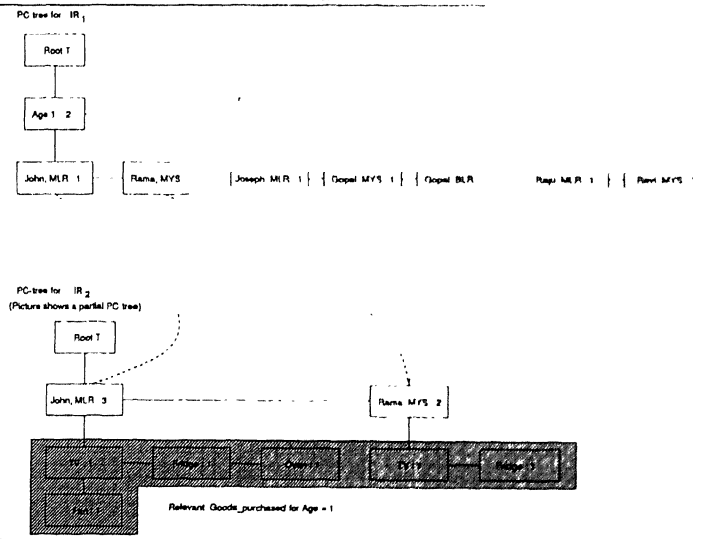


Figure 6.16 PC-trees with the linking mechanism



6. **Mining semantic partition for frequent set of values**

INPUT : Linked PC-trees

OUTPUT : Blocks B_1, B_2, ... B_ρ in the semantic partition of $\mathbb{R}$

STEPS :

Each PC-tree has two sets of nodes corresponding to the values of characteristic attribute/common set of attributes. Let the first set of nodes for each PC-tree, PC_x be N_1^x, N_2^x, ..., N_{xA}^i and the second set of nodes, which are the children nodes of the first set of nodes be C_1^x, C_2^x, ..., C_{xB}^x. [Note : for PC_1, $xA = 1A = \rho$.]

For each N_j^i, $(1 \leq i \leq m,\ 1 \leq j \leq iA)$ do the following :

(a) Let the child nodes of N_i^i be $C_1^i, C_2^i, \ldots C_{iB}^i$.

(b) For each C_k^i, $(1 \leq i < m,\ 1 \leq k \leq iB)$ do the following :

(c) Identify the set of nodes, $N_1^{i+1}, N_2^{i+2}, \ldots, N_{iA}$ whose *item-name* fields are same as *item-name* field of C_k^i. Let the child nodes of N_j^{i+1} $(1 \leq j \leq i\acute{A})$ be C_l^{i+1} $(1 \leq l \leq i\acute{B})$.

(d) Repeat steps 6b and 6c till we identified the child nodes (which may be a tree) of the PC-tree, PC_m corresponding to N_j^i.

(e) The count value of each identified child node, C_x^m and its child nodes which follows it is calculated and updated by multiplying count values of each child nodes by the count value pertaining to the child nodes C_z^y $(1 \leq y < m)$ which are on the path from PC_1 to PC_m leading to C_x^m.

(f) For each N_i^1, the identified collection of child nodes, C_x^m $(1 \leq x \leq r)$ represents a block B_i of the *semantic partition*.

(g) Mine B_i, which is a part of the PC-tree, PC_m as discussed in section 6.2.5.

In Figure 6.16 the marked area represent the block B_1 correspond to the node, where $Age = 1$. It is a part of the PC-tree to be mined for frequent set of values.

Experiment pertaining to mining multiple databases is given in next section.

6.6 Experimental Results

In this section, we report results on experiments conducted using the tree structures, viz., PC-tree, GPC-tree, MPC-tree, LOFP-tree, GLOFP-tree, and MLOFP-tree. We also report the efficiency of distributed structure generation algorithm based on PC-trees, PC-tree based dynamic mining, and PC-tree based mining of multiple databases.

We used the synthetic data generator given in [Quest site] to generate the transaction databases. The parameters used are shown in Table 6.3 and their settings in the generation of the synthetic data are shown in Table 6.4.

Table 6.3 Parameters

Parameters	Descriptions
$\|D\|$	Number of transactions
$\|T\|$	Average size of transactions
$\|I\|$	Average size of maximal potentially frequent itemsets
$\|L\|$	Number of maximal potential frequent itemsets
NOI	Number of items
σ	User specified minimum support, 0.75%

Table 6.4 Parameters settings

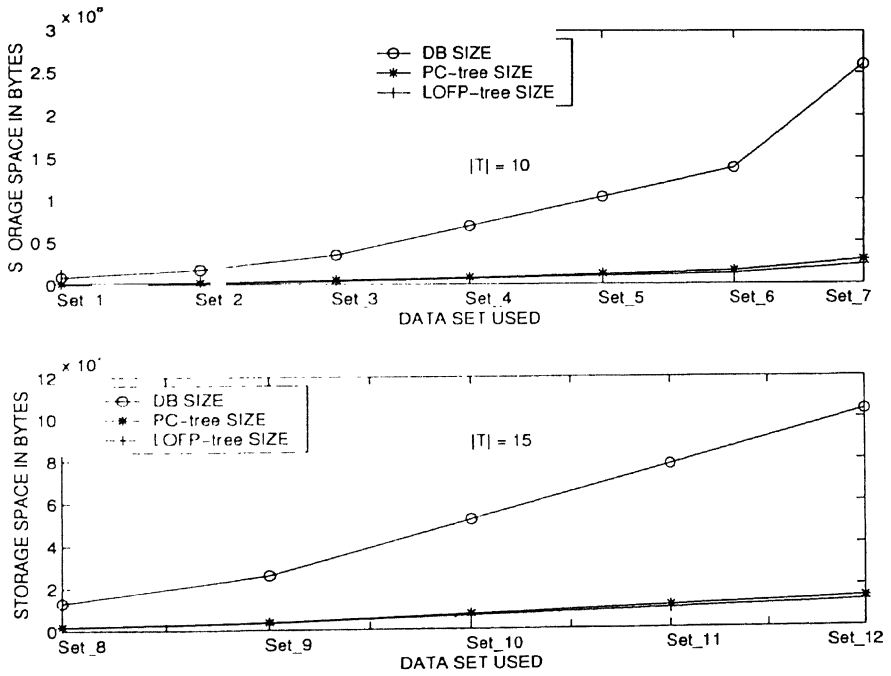
Name	$\|D\|$	$\|T\|$	$\|I\|$	$\|L\|$	NOI	Name	$\|D\|$	$\|T\|$	$\|I\|$	$\|L\|$	NOI
Set_1	25K	10	4	1000	1000	Set_7	800K	10	4	1000	1000
Set_2	50K	10	4	1000	1000	Set_8	25K	15	4	1000	1000
Set_3	100K	10	4	1000	1000	Set_9	50K	15	4	1000	1000
Set_4	200K	10	4	1000	1000	Set_10	100K	15	4	1000	1000
Set_5	300K	10	4	1000	1000	Set_11	150K	15	4	1000	1000
Set_6	400K	10	4	1000	1000	Set_12	200K	15	4	1000	1000

Experiment 1 : Memory requirements

In this experiment, we constructed the PC-tree, based on the algorithm given in section 6.2.1.2, and the LOFP-tree given in section 6.2.2 for the datasets Set_1 to Set_12. Figure 6.17 shows the memory requirements for database, PC-tree and

LOFP-tree for the datasets Set_1 to Set_12 given in Table 6.4. It is clear from the graphs that even though the size of PC-tree is greater than that of the LOFP-tree, the memory size requirement of PC-tree increases nominally with a significant increase in the database size. For example, the ratio of the storage space for the database to that of PC-tree keeps increasing from 7 to 10 as the database size increases from 8.5MB to 260MB, where $|T| = 10$. [i.e., as the database size increases from 8.5MB to 260MB, size of the PC-tree with respect to that of database is decreases from 13% to 10%]. So, PC-tree requires a *lesser* memory space than the database. The experiment also shows that there is no significant difference between

Figure 6.17 Comparison based on the space requirements



PC-tree size and LOFP-tree size. The dynamic mining property of the PC-tree is an important contribution in the sense that size of PC-tree is nearly equal to that of the LOFP-tree, but PC-tree is a complete representation of the database and is amenable for dynamic mining.

The total number of disk block accesses for generating frequent itemsets using PC-tree is less than that for the LOFP-tree. For example, consider the data sets Set_3 and Set_4. Table 6.5 shows the number of disk block accesses for both the structures. We assumed that each disk block size = $2K$ Bytes and main memory size = 3000 Blocks ($6000K$ Bytes).

Table 6.5 Total number of disk block accesses for generating frequent itemsets using PC-tree and LOFP-tree

	Set_3	Set_4
PC-tree Size	4872176 Bytes = 2379 Disk blocks	7978611 Bytes = 3896 Disk Blocks
LOFP-tree Size	3355998 Bytes = 1639 Disk blocks	6367898 Bytes = 3110 Disk blocks
DB Size	34414205 Bytes = 16804 Disk blocks	68949580 Bytes = 33667 Disk blocks
# of Disk block accesses for PC-tree	$16804 + 0 = 16804$	$33667 + (3896 - 3000) \times 2 =$ 35459
# of Disk block accesses for LOFP-tree	$2 \times 16804 + 0 = 33608$	$2 \times 33667 + (3110 - 3000) \times 2$ = 67554

From the fifth and sixth rows of Table 6.5, it is clear that total number of disk block accesses for construction and storage of PC-tree is lesser than that of LOFP-tree; with respect to Set_3, 50% reduction in number of disk block accesses and with respect to Set_4, it is 47.5%.

Experiment 2 : Timing requirements

We conducted an experiment using the data sets Set_1, Set_2, Set_3 and Set_4 to compare the time required to construct the LOFP-tree directly from the database which requires two database scans and also from the PC-tree, which requires one database scan and one scan of the PC-tree. Table 6.6 shows the values obtained. It can be seen that time required to construct the LOFP-tree based on PC-tree is lesser than that of its construction directly from the database. These results indicate that the time required to construct LOFP-tree from the PC-tree is smaller than that of direct construction of LOFP-tree by 2 to 22 sec.

Table 6.6 Timing requirements to construct LOFP-tree : A comparison

Data Set	Set_1	Set_2	Set_3	Set_4
Time to construct LOFP-tree from PC-tree	22 sec.	45 sec.	88 sec.	173 sec.
Time to construct LOFP-tree directly from database	24 sec.	48 sec.	97 sec.	195 sec.

In **Experiment 3** and **Experiment 4**, we show the memory requirements of MPC-tree, MLOFP-tree, GPC-tree and GLOFP-tree. For this we use data sets Set_3 and Set_4 for illustration.

- **Experiment 3 : Memory requirements for modified tree structures**

In this experiment, we generated MPC-tree and MLOFP-tree by creating frequent 2-itemsets in one scan. For this, we used an upper triangular matrix (UTM) to store the 2-itemsets obtained during the first scan as discussed in subsection 6.2.4. In the second scan we constructed MPC-tree and MLOFP-tree based on the frequent 2-itemsets. The sizes of the MPC-tree and MLOFP-tree are smaller than that of the LOFP-tree as shown in Table 6.7.

Table 6.7 LOFP-tree size Vs. MLOFP-tree size and MPC-tree size

	Set_3	Set_4
MLOFP-tree Size	486343 Bytes	833406 Bytes
MPC-tree Size	525614 Bytes	899416 Bytes
LOFP-tree Size	3355998 Bytes	6367898 Bytes

Even though the storage space for MLOFP-tree and MPC-tree is less than that of LOFP-tree, the cost paid for this is in terms of additional storage space for storing 2-itemsets (in terms of UTM). If we consider 2 bytes per item number, space requirement for UTM is $1001 \times 500 \times 2 = 1001000$ Bytes, when compared with the storage space for 1000 item numbers; which is $1000 \times 2 = 2000$ Bytes to store frequent 1-itemsets. However, in this case the memory requirement for the MPC-tree for Set_3 is $1001000 + 525614 = 1526614$ Bytes and for the MLOFP-tree it is $1001000 + 486343 = 1487343$ Bytes. But, for LOFP-tree, it is $2000 +$

3355998 = 3357998 Bytes. So, the total memory required to construct the MLOFP-tree and MPC-tree is smaller than that of the LOFP-tree. Note that, for storing frequent 3-itemsets corresponding to Set_3, we require 333335000 Bytes to store 3-itemsets, which itself is larger than the overall memory requirement for constructing LOFP-tree or MPC/MLOFP-tree. This prompted us not to consider generation of PC/LOFP-trees based on frequent 3-itemsets. If we consider the possible frequent 3-itemsets arising out of frequent 2-itemsets, the size of MPC/MLOFP-tree may be reduced further.

A similar improvement, in terms of memory requirement is exhibited by the MPC/MLOFP-tree over the LOFP-tree corresponding to Set_4 as shown in the third column of Table 6.7.

Experiment 4 : Memory requirements for generalized tree structures

We conducted an experiment to construct generalized PC-tree (GPC-tree) and generalized LOFP-tree (GLOFP-tree). To construct these tree structures, we generalized the items based on an *is_a* hierarchy. In our experiment, we generalized the items as shown in Table 6.8 which is based on the *is_a* hierarchy shown in Figure 6.5A. Due to the generalization, the frequent patterns are increased and the

Table 6.8 Generalization of items

Item no.	0 - 99	900 - 999
Generalized item no.	0	9

sizes of GPC-tree and GLOFP-tree are reduced. For Set_1, GPC-tree size is reduced by a factor of 572.89 with respect to LOFP-tree and GLOFP-tree size is reduced by a factor of 582.44 with respect to LOFP-tree. For Set_4, GPC-tree size is reduced by a factor of 1032.07 with respect to LOFP-tree and GLOFP-tree size is reduced by a factor of 1047.52. Table 6.9 shows the sizes of LOFP-tree, GPC-tree and GLOFP-tree.

Table 6.9 LOFP-tree size Vs. GPC-tree size and GLOFP-tree

	Set_3	Set_4
LOFP-tree Size	3355998 Bytes	6367898 Bytes
GPC-tree Size	5858 Bytes	6170 Bytes
GLOFP-tree Size	5762 Bytes	6079 Bytes

The method followed to construct GPC/GLOFP-tree is discussed in subsection 6.2.3. For each transaction, we generalized the item number to the appropriate one of the values in the range 0 to 9 using Table 6.8. We removed the duplicates since we concentrate only on presence or absence of item numbers. We built the GPC-tree and GLOFP-tree based on this generalized transactions. Since the sizes of GPC/GLOFP-trees are significantly smaller than the size of the main memory, one can construct a memory resident GPC/GLOFP-tree even for very large transaction databases (say, size of the database is of the order of Terra bytes).

Even though the GLOFP-tree occupies less memory space than that of GPC-tree, the main difference between GLOFP-tree and GPC-tree is that GPC-tree can be constructed with **single database scan** whereas we need **at least two database scans** to construct the GLOFP-tree.

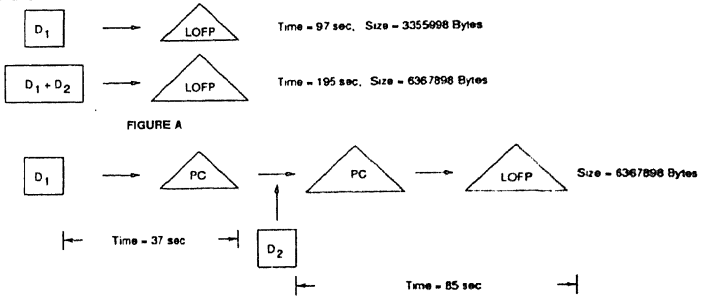
- Experiment 5 : Dynamic mining

The efficiency of mining a transaction database for association rules using FP-tree (LOFP-tree) is already established [Han et.al, 00]. So, we show the dynamic mining capability of PC-tree based scheme in constructing LOFP-tree and its variant, GLOFP-tree efficiently as target structures. We consider each requirements of dynamic mining separately, as follows :

1. Change of data :

   We conducted an experiment using the dataset, Set_4. We constructed a LOFP-tree directly using a portion (50%) of Set_4 (say, D_1). Later, we added the remaining 50% (say, D_2) to D_1 (i.e., $D_1 + D_2$ = Set_4). In each case, LOFP-tree has to be constructed separately using the algorithm given in [Han

Figure 6.18 Handling of change of data



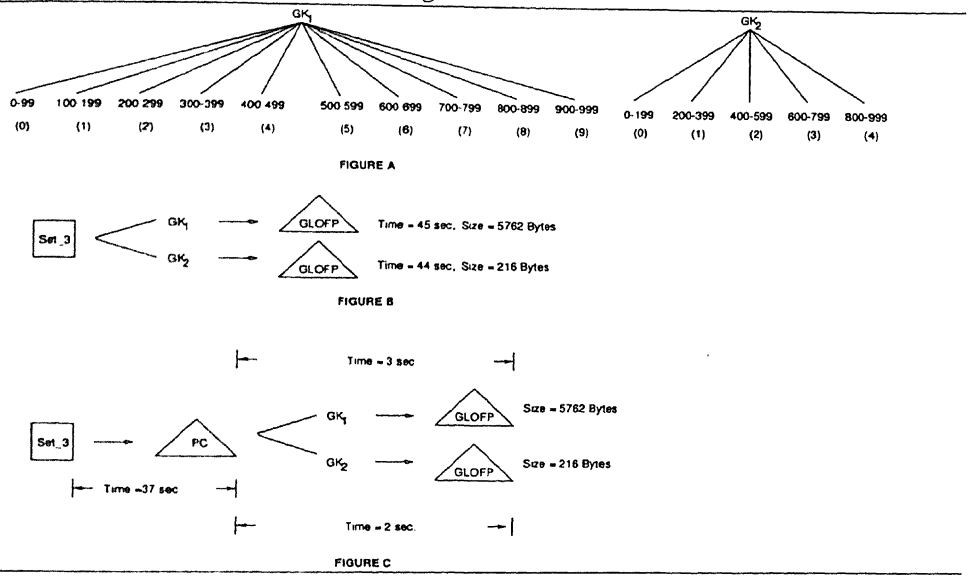
et.al, 00], since LOFP-tree based scheme does not handle incremental change of the database. The timing requirements in these two cases are 97 sec. and 195 sec. respectively. This is shown pictorially in Figure 6.18A. We, then, constructed LOFP-tree indirectly as follows : first, a PC-tree is constructed using D_1 and then, that PC-tree is incrementally updated using D_2. LOFP-tree is generated based on the updated PC-tree. The time required to construct the PC-tree from D_1 is 37 sec., and the time required to update the PC-tree using D_2 and build the LOFP-tree from the updated PC-tree is 85 sec. This is shown pictorially in Figure 6.18B.

So, incremental construction of LOFP-tree using PC-tree as an intermediate representation is better than that of its direct construction.

2. Change of knowledge :

In this experiment, we used two knowledge trees shown in Figure 6.19A as generalization trees which represent the generalization of items. GK_1 is an initial knowledge tree and GK_2 represents a variant of GK_1. Dataset used is Set_3. We show in this experiment how the dynamic mining capability of PC-tree based scheme can be exploited in constructing GLOFP-trees efficiently. Since LOFP-tree based scheme does not handle change of knowledge without referring to the original database, for each knowledge tree, Set_3 has to be referred. The time required to construct GLOFP-tree using Set_3 and GK_1 is

Figure 6.19 Handling of change of knowledge



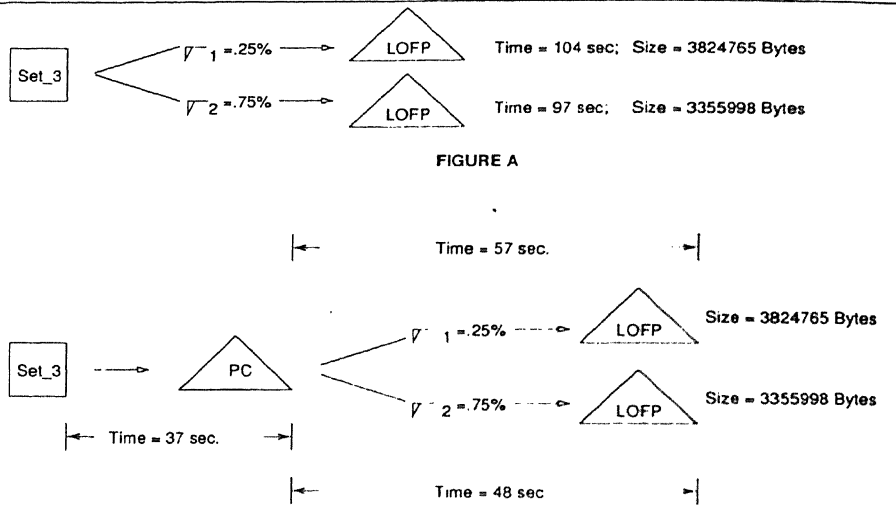
45 sec. and the time required to construct GLOFP-tree using Set_3 and GK_2 is 44 sec. This is shown pictorially in Figure 6.19B. We then constructed PC-tree using Set_3. Using this PC-tree and GK_1 and GK_2, we independently generated GLOFP-trees without referring to the dataset, Set_3. The time required to construct PC-tree from Set_3 is 37 sec. and time required to generate GLOFP-tree using GK_1 and PC-tree is only 3 sec., and the time required to generate GLOFP-tree using GK_2 and PC-tree is only 2 sec. This is pictorially shown in Figure 6.19C.

This result shows the efficient incremental construction of GLOFP-trees using PC-tree as the intermediate representation.

3. Change of user defined minimum support value :

In this experiment we used two user defined minimum support values, σ_1 = 0.25% and σ_2 = 0.75%. Dataset used is Set_3. Since LOFP-tree based scheme can not handle change of user defined minimum support value without referring to the original database, for each user defined minimum support value, we need to refer to the dataset, Set_3 to construct the LOFP-tree. The

Figure 6.20 Handling of change of user defined minimum support value



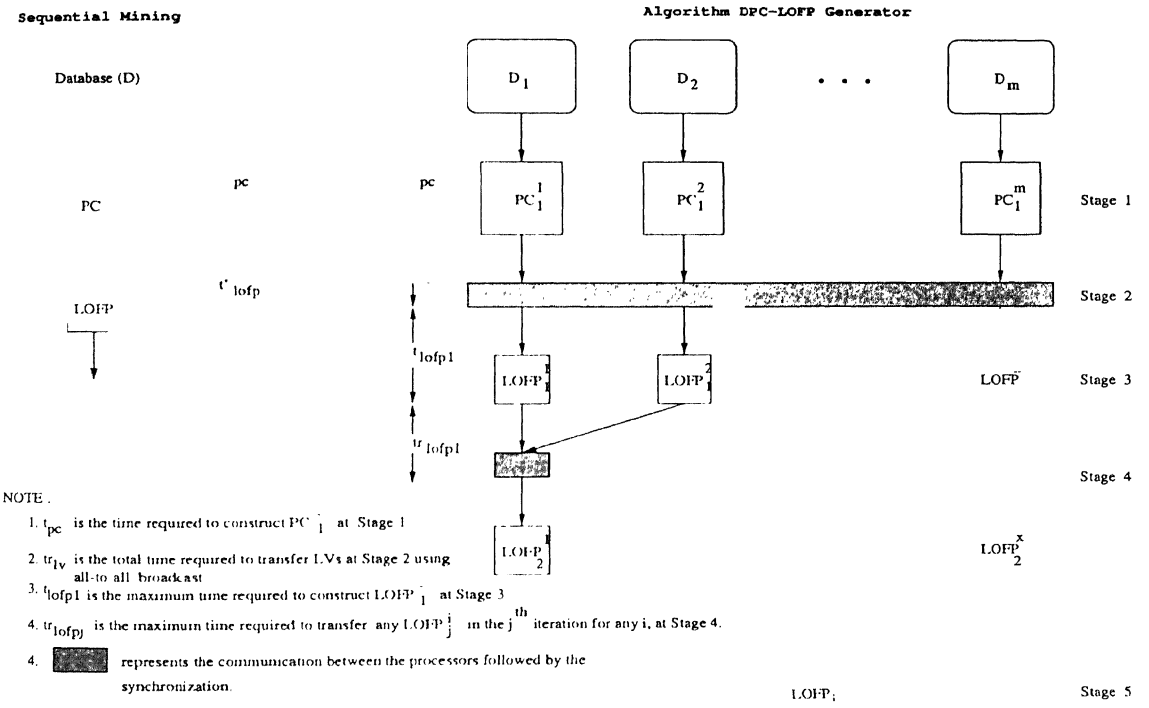
time required to construct LOFP-tree using Set_3 and σ_1 is 104 sec. and the time required to construct LOFP-tree using Set_3 and σ_2 is 97 sec. This is shown pictorially in Figure 6.20A. On the other hand, the time required to construct PC-tree from Set_3 is 37 sec. and the time required to generate LOFP-tree pertaining to σ_1 is 57 sec. and the time required to generate LOFP-tree pertaining to σ_2 is 48 sec. This is shown pictorially in Figure 6.20B.

This experiment reveals the efficiency of the PC-tree based scheme for constructing LOFP-tree based on changing user defined minimum support values.

Experiment 6 : Efficiency of distributed structure generation algorithm (DPC-LOFPG)

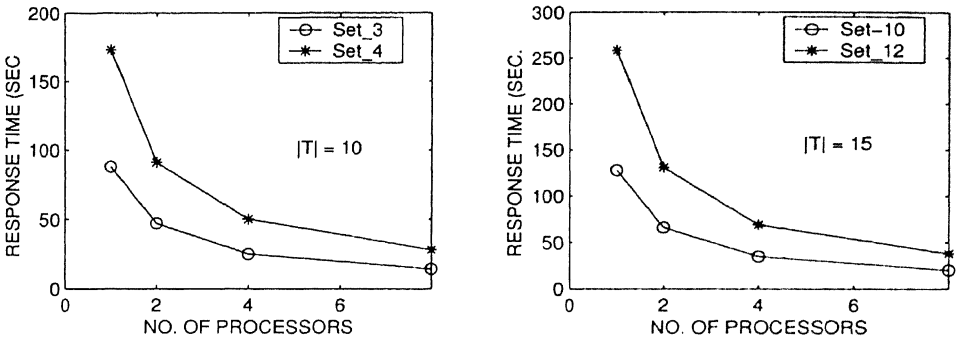
We compared **DPC-LOFPG** algorithm with its sequential counterpart. Figure 6.21 shows the block schematic diagram that depicts the stages with the corresponding timing requirements. From the block schematic shown in Figure 6.21, the time required to construct LOFP-tree from sequential mining, T and time for constructing global LOFP-tree from DPC-LOFPG algorithm, S are given by the following formulas : $S = t_{pc} + tr_{lv} + \{t_{lofp1} + tr_{lofp1} + t_{lofp2} + tr_{lofp2} + \cdots + t_{lofpj} + tr_{lofpj}\}$,

Figure 6.21 Timing requirements for DPC-LOFPG and its sequential counterpart



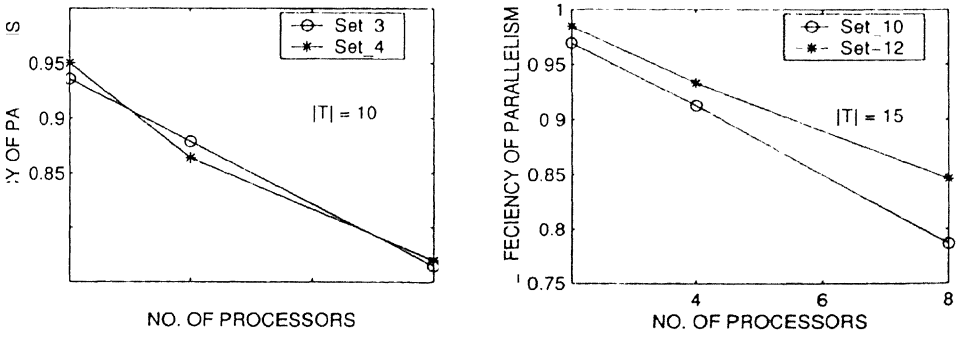
if no. of processors $= 2^j$ at Stage 1, for j > 0. Corresponding to S, we have $T = t'_{pc} + t'_{lofp}$. To study the efficiency and response time of the parallel algorithm **DPC-LOFPG**, corresponding to its sequential counterpart, we conducted a simulation study by varying the number of processors from 1 to 8. Data sets used in our experiment are Set_3, Set_4, Set_10 and Set_12. Figure 6.22 shows the response time for different number of processors. *Efficiency of parallelism* is defined

Figure 6.22 Response time



as $T/(S \times m)$, where S is the response time of the system when m (> 1) number processors are used; and T is the response time when $m = 1$. We show in Figure 6.23 the efficiency of parallelism for different number of processors. It may be observed

Figure 6.23 Efficiency of Parallelism



from Figure 6.23 that the best value of efficiency is 98% for Set_12 and 95% for Set_4, both are exhibited by the 2-processor system.

Experiment 7 : Mining multiple databases for associations

We used three databases for the experimental study and conducted an experiment as discussed in section 4.6. However, instead of generating EIF as the intermediate structure, here, we created three PC-trees. Each PC-tree has two sets of nodes. For the first PC-tree, the first set of nodes correspond to the generalized values of attribute *Age* and the second set of nodes correspond to the values of attribute *SSN*. For the second PC-tree, they correspond to the values of attributes *Sec_no* and *CCN*; and for the third PC-tree, they correspond to the values of attributes *Card_no* and $Goods\_purchased_i$ ($0 \leq i \leq 999$). The number of frequent tuples generated, number of association rules and size of each block (in terms of number of tuples) are same as in Table 4.3. Our experiment shows that total size of all PC-trees which are used for multi-database mining is 48% of the size of EIF, and the size is 37422160 Bytes. The advantage of multiple PC-tree based mining for association rule generation over the EIF based scheme are:

1. After the *Trimming* process (refer to step 3 of Algorithm **MMUP** in section 6.5), creation of PC-trees needs *only one* scan of each of the trimmed relations. However, the number of scans on the relation to construct the EIF from trimmed relations depends on the generalized values of the selected attributes in each trimmed relation.

2. Change of data in the selected relation prompts to change only in the corresponding PC-tree, so the approach is modular. But the change of data in the selected relation needs to re-create the EIF structure.

3. Our experimental study reveals that the size of intermediate structure (i.e., PC-trees) generated by **MMUP** algorithm is *smaller* than EIF structure.

6.7 Usage of PC-tree for Other Data Mining Applications

We discuss PC-tree based pattern classification [Duda et.al, 73] in section 6.7.1 and PC-tree based clustering [Jain et.al, 99] in section 6.7.2

6.7.1 Classification

PC-tree can be used to solve pattern classification problem. The PC-tree is constructed using labelled training data and then, the test data is checked against each path of PC-tree to determine its class label. The example below shows how the training patterns can be stored in a PC-tree.

Example : Let us consider the patterns for digits '0' and '9' shown in Figure 6.24A. Each pattern is a matrix of size 4×4. A '1' in a location indicates the presence and a '0' indicates the absence of the feature. Figure 6.24B gives an equivalent representation of the patterns using features that are present. Features are numbered row wise. Figure 6.24C shows the PC-tree for the corresponding data patterns based on class labels followed by the features present.

Figure 6.24 Data patterns for '0' and '9'

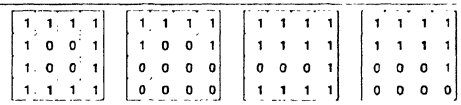


Figure A : Patterns in 4 X 4 matrix form

Pattern #	Class label	Features present
1	0	1, 2, 3, 4, 5, 8, 9, 12, 13, 14, 15, 16
2	0	1, 2, 3, 4 5, 8
3	9	1, 2, 3 4, 5, 6, 7, 8, 12, 13, 14, 15, 16
4	9	1, 2, 3 4, 5, 6, 7, 8, 12

Figure B Representation of patterns in Figure A

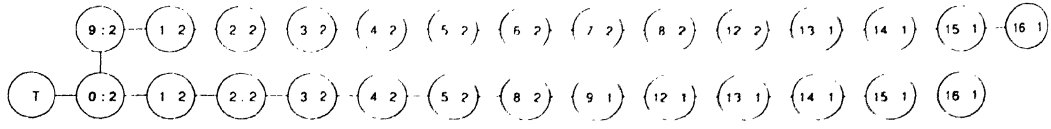


Figure C : PC-tree to hold the data patterns

Note : In Figure C, horizontal link represents c-pointer and vertical link represent s pointer Each node has two parts Item name and Count value

Note that even though there are four tuples in Table 6.24B, the corresponding PC-tree (Figure 6.24C) has only two branches. This property of the PC-tree will be

helpful in following ways:

1. It reduces the number of training patterns to be considered for classifying the test patterns which in turn reduces space and time requirements.
2. It decreases improper classification of test patterns by not considering the patterns like Pattern#2 and Pattern#4 which are totally subsumed by Pattern#1 and Pattern#3 respectively.

We give the classification algorithm based on PC-tree below :

Classification algorithm based on PC-tree (PC-Classifier)

INPUTS :

1. T_S - Test dataset (A collection of test patterns.)
2. T_R - Training dataset (A collection of training patterns.)

OUTPUTS :

1. C_{time} - Classification time.
2. CA - Classification accuracy.

STEPS :

1. Generate PC-tree using T_R (refer section 6.2.1.2 for details).
2. *start-time* = time().
3. For each $s_i \in T_S$
   (a) Find n_i, set of positions of non-zero values corresponding to s_i.
   (b) Find the nearest neighbour branch, e_k in PC-tree depending on maximum number of features which are common to both e_k and n_i.

(c) Attach the label l associated with e_k to n_i.

(d) If (l == label of s_i) then

$$correct = correct + 1$$

4. *end-time* = time().

5. $CA = \frac{correct}{|T_S|} \times 100$ // $|T_S|$ is the number of test patterns.

6. Output C_{time} = *end-time - start-time.*

7. Output CA.

We conducted experiments using handwritten digit data [Prakash et.al, 97]. The training dataset consists of 667 patterns for each class of digits labelled from 0 to 9, totalling to 6670 patterns. Each pattern is having 192 features. The test dataset consists of 3333 patterns. The classification accuracy using **PC-Classifier** is 93.61% [Ananthanarayana et.al, 01C] which outperforms the other classifiers including *k-NNC* based on the results reported in the literature [Prakash et.al, 97].

6.7.2 Clustering

PC-tree can be used for clustering the data in a direct way [Ananthanarayana, et.al, 01B]. Here, *cluster description* is the set of features along the path from the root to the leaf node of PC-tree. All the patterns that are mapped onto the path form the members of the cluster. PC-tree requires less space to hold the clusters because of two reasons : (i) if two or more patterns share a prefix, P, then P is stored *only once.* (ii) only non-zero positional values are stored in the PC-tree.

PC-tree based clustering algorithm (PC-Cluster)

STORE :

For each pattern, $P_i \in D$ (database) :

If any prefix sub pattern, SP_i, of P_i exists as a prefix in a branch e_b :

Put features in SP_i in e_b, by incrementing the corresponding count field value of the nodes in the PC-tree. Put the sub patterns, if any, of P_i by appending additional nodes with count field value equal to 1 to the path in e_b.

Else Put P_i as a new branch of the PC-tree.

RETRIEVE :

For each branch, $B_i \in PC$ (PC-tree) :

For each node, $N_j \in B_i$:

Output N_j.Feature.

The features corresponding to the nodes in the branch of the PC-tree from root to leaf constitute a prototype.

For example, consider the patterns for the digit '1' and '7' as shown in Figure 6.25A. Each pattern is a matrix of size 4×4. A '1' in a location indicates the presence and a '0' indicates the absence of the feature. Figure 2.1B gives an equivalent representation of the patterns using features that are present. Figure 2.1C shows the PC-tree for the corresponding data patterns. Here, the cluster descriptions are {3,7,11,14,15,16,}, {3,7,11,15}, {1,2,3,7,11,15}. In the case of PC-tree, the cluster description itself forms the prototype. Members of the cluster, with description {3,7,11,14,15,16} are {3,7,11,14,15,16}, {3,7,11,14,15} and {3,7,11}.

We conducted experiments using handwritten digit data [Prakash et.al, 97]. We compared our clustering algorithm, **PC-Cluster**, with respect to the popular single scan clustering algorithm, *Leader* [Spath 80] based on two parameters [Ananthanarayana et.al, 01B]. They are (i) the time required to obtain the clusters and (ii) the space required to hold the clusters. **PC-Cluster** took 62% of the time of *Leader* to construct the clusters and

Figure 6.25 Data patterns for '1' and '7'

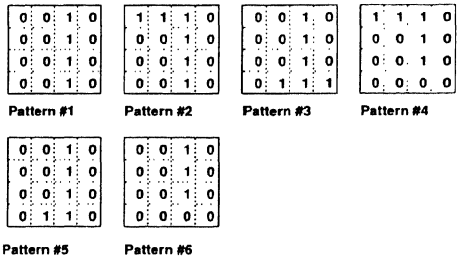


Figure A : Patterns in 4 X 4 matrix form

Pattern #	Class label	Features present
		3,7,11,15
		1,2,3,7,11,15
		3,7,11,14,15,16
		1,2,3,7,11
		3,7,11,14,15
		3,7,11

Figure B : Representation of patterns in Figure A

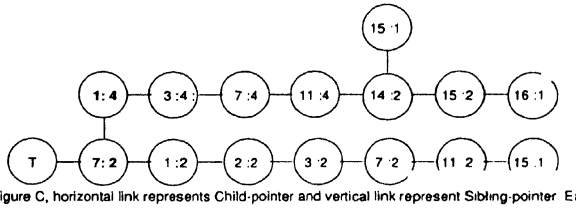


Figure C : PC-tree to hold the data patterns

Note : In Figure C, horizontal link represents Child-pointer and vertical link represent Sibling-pointer Each node has two parts Item-name and Count value

the clusters generated by **PC-Cluster** occupied 31% of the space occupied by the clusters generated by *Leader*. The advantage of the proposed cluster scheme, **PC-Cluster** [Ananthanarayana et.al, 01B] are : (i) cluster representations can be obtained using *a single database scan* and (ii) space to hold clusters and time required to generate clusters are less when compared with other single pass clustering algorithms like *Leader*.

6.8 Summary

- We presented a **novel** data structure called PC-tree. We showed that PC-tree is a **complete** and **compact** representation of the transaction database. This structure is constructed using **a single database scan**.

- We presented variants of PC-tree and LOFP-tree called MPC-tree and MLOFP-tree respectively based on frequent 2-itemsets. **Two database scans** are required to construct either the MPC-tree or the MLOFP-tree. We also presented the construction of Generalized PC-tree (GPC-tree) and Generalized FP-tree (GLOFP-tree). Both representations need less storage space when compared with FP-tree.

Moreover, construction of GPC-tree needs only **one database scan** where as construction of GLOFP-tree needs **two database scans.**

- We introduced the notion of **dynamic mining** that is a significant extension of incremental mining and we have shown that PC-tree is *ideally* suited for dynamic mining.
- We proposed a distributed structure generation algorithm for mining frequent itemsets based on the PC-tree structure.
- We discussed the scheme for mining multiple databases using a collection of PC-trees.
- We proposed schemes based on PC-tree for efficient pattern classification and clustering.

Based on the experimental results we have the following conclusions :

- For generating association rules, the memory requirements to store the PC-tree structure will increase nominally as the database size increases significantly and the database is scanned **only once.**
- Number of disk block accesses required to generate frequent itemsets using PC-tree is **lesser** with a reduction upto 50% of that of LOFP-tree.
- LOFP-tree can be built efficiently in terms of construction time using PC-tree as an intermediate structure when compared with direct construction of LOFP-tree from the database.
- For mining association rules from very large databases, MPC-tree and MLOFP-tree, which are based on frequent 2-itemsets **seem to be the best** in terms of number of candidate itemsets explored.
- Mining frequent itemsets using MPC/MLOFP-tree, constructed based on frequent p-itemsets, where $p > 2$, seems to require a very huge memory space. So, it appears that construction of MPC/MLOFP-tree based on frequent 2-itemsets is adequate.

It is possible to successfully exploit the structure of the PC-tree to generate generalized associations using **a single database scan**. This is because, the number of nodes in the corresponding GPC-tree will increase nominally with the size of transaction database.

Dynamic mining capability of PC-tree based scheme is exploited in efficient incremental construction of LOFP-tree and GLOFP-tree.

The proposed distributed algorithm, DPC-LOFPG, is found to be efficient because it scans the database *only once* and it does not generate any candidate itemsets. Further, DPC-LOFPG exploits parallelism to a large extent and the best value for efficiency of parallelism is about 98%.

Total size of all PC-trees which are used for mining multiple databases is 48% of the size of EIF.

Chapter 7

Association Generation Using Multi-database Domain Link Network

7.1 Introduction

In this chapter, we propose a scheme which uses knowledge in the form of semantic network and a collection of databases as input [cf. 3.4.1] for discovering associations.

The A O taxonomy based scheme discussed in chapter 5 uses explicitly generated valuetable for mining multiple databases for associations. In chapter 4 and chapter 6, we proposed schemes, AMAAM and MMUP respectively, for mining multiple databases for association rules where valuetable is generated implicitly. The scheme which generates valuetable implicitly employ knowledge in the form of a *semantic network* to obtain the navigation path across relevant databases. In chapter 4, an intermediate structure, called *extended inverted file* (EIF) is used to find associations between the values of attributes which may belong to different databases. The drawbacks of this scheme are :

D1 : Number of scans on the selected relations to generate structure (i.e., EIF) depends on generalized values of the selected attributes in each of the relations of the databases.

D2 : One of the inputs to the **AMAAM** algorithm is the indication of antecedent and consequent characteristic concepts and hence the characteristic attributes, between the values of which the associations has to be discovered. This constrains the relations and hence the structure. Later, if a user wishes to change the indication of antecedent and consequent or to find the associations between some other attributes, then even though they exist in the input databases, there is a need to re-execute **AMAAM** which may be a costly affair because of number of database scans involved.

D3 : Not amenable for incremental structure generation. That is, once the structure is constructed, *change of data* in any relation based on which the structure is built, leads to reconstruction of the entire structure.

D4 : Selection of relevant relations and attributes by the navigation algorithm (i.e., relation of reference) before creating the structure may need multiple scans of the databases.

Some of these problems (**D1** and **D3**) are solved by **MMUP** scheme discussed in chapter 6. They are :

1. Construction of the structure (i.e., a collection of PC-trees) needs to scan each identified relations *only once*.

2. Once the attributes are identified in the selected relation, then each PC-tree can be built independently. So, this scheme has two benefits.

   (a) Structure can be updated incrementally. That is, *change of data* in any one of the selected relation needs to update a branch or a couple of branches in a PC-tree.

   (b) The component PC-tree of the structure can be generated in parallel. That is, the structure can be built using multiple processors. This is more efficient because there is no need for synchronization between the processors during construction of PC-trees.

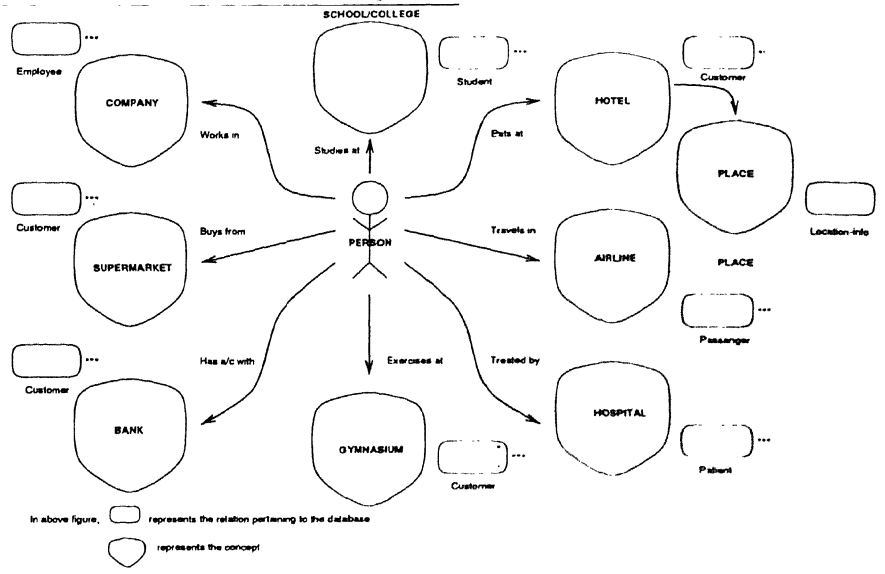
3. Structures generated by this scheme are more *compact* than the structure generated by **AMAAM** scheme. This is due to the constructional procedure of PC-tree.

However, drawbacks **D2** and **D4** of **AMAAM** still exist in **MMUP** scheme. This is because it uses the same navigation method used by **AMAAM**.

In this chapter, we propose a scheme that generates a structure which is more *compact* than PC-tree and overcomes the drawbacks of both the schemes discussed above.

All existing association mining schemes including the schemes proposed by us in previous chapters, find the associations based on explicit co-occurrence of values. In the case of schemes which are based on single relation, the co-occurrence of values is direct. In the case of schemes which are based on multiple relations (eg. the schemes - **AMAAM** and MMUP- proposed in chapter 4 and chapter 6 respectively), structures are generated in such a way that the co-occurrence of values are preserved. For example, in finding the associations between the values of person's *age* and *magazine* s/he reads, attribute *age* and *magazine* may not appear in the same relation. In such cases, the scheme proposed in previous chapters for mining multiple databases, we use the notion of common set of attributes to preserve the co-occurrences while building the structures. In this chapter, we show that by indirectly preserving the co-occurrence of values it is possible to mine association rules. This can be achieved by using the notion of dominant entity. We illustrate this notion with the following example. One can depict a person oriented information system as shown in Figure 7.1. Here, CUSTOMER database describes PERSON's service entertainment behaviour; EMPLOYEE database gives his professional information; PATIENT database gives his medical history; and AIRLINE database gives PERSON's travel details. There could be an implicit association among various parts of this data. There may be an association between salary, region of living, mode of traveling and disease; like *people who have salary in the range of US$ 10,000 - US$ 20,000, eat frequently at CITY X hotels and travel by air in executive class have cardiovascular diseases*, where CITY X is marked with a high pollution rating. This example provides the associations of a person in different contexts. We call such a mining activity **person-based association mining** (or **personalized mining**) and the system under which the mining activity is

Figure 7.1 Person oriented information system



performed is called **person warehouse**. The primary property of such a warehouse is the possibility to generate a global schema linked by one entity which we call, *dominant entity*. The associations between the values of attributes of interest using the dominant entity are called **dominant entity based associations**. We propose and describe an efficient structure to perform this mining activity. In the above example, the dominant entity is *person* which is characterized by attributes *name and address*. If we consider the academic information system, the dominant entity could be *student* which is characterized by attribute *student-id*. Here, we are addressing the problem of 'dominant entity based association mining activity in multiple databases'. For illustration purposes we use 'person warehouse' through out the chapter. However, the notion can suit any domain where we have a *dominant entity* and the data warehouse.

The contributions of this chapter are :

To propose a scheme for discovering associations among attribute values across multiple databases using notion of the dominant entity.

Generation of a novel data structure for linking domains of attributes which belong to multiple databases. This aids in finding associations ***simultaneously*** *between*

the values of attributes.

- *Discovery of associations using a novel data structure, which needs* ***a single scan*** *of each of the relevant databases.*

- *To propose and use a structure that is* ***compact*** *with respect to multiple databases among all proposed schemes; and the mining algorithm based on the proposed structure is* ***modular****.*

- *Proposal of a scheme called* dominant entity based association rule mining, *which is not* ***constrained to look for co-occurrence*** *of values in tuples; and efficient discovery of these association rules using the proposed structure.*

7.2 Problem Definition

In this work, we develop a structure for mining several databases using a dominant entity. Our aim is to find all possible interesting associations. In general, these associations may be viewed as *n-ary* associations (where, $n \geq 2$) between values of attributes belonging to different relations/databases. So, we assume, without loss of generality, that schemas of relations under consideration are having at least one characteristic attribute which is involved in the association generation process.

An *n-ary* association gives a relationship between set of values of n characteristic attributes, $CA_{i1}^{j1}, CA_{i2}^{j2}, \ldots CA_{in}^{jl}$ where, each of the characteristic attributes may belong to different relations/databases.

Definition 7.2.1: Dominant Entity Attribute(s) (DEA): It is a collection of attributes which *characterizes* the dominant entity. Its values are unique and hence can be used to navigate between the values of characteristic attributes which belong to the same relation or different relations of same or different databases.

For example, in the person information system, 'person' is the dominant entity. Each relation schema under consideration has attribute pair <name, address> corresponding to the entity 'person', and the attribute collection is called DEA.

The conventional association rule, $\{x_1, x_2, \ldots, x_n\} \Longrightarrow \{y_1, y_2, \ldots, y_m\}$ (where $x_1, x_2, \ldots x_n, y_1, y_2, \ldots y_m$ are values) holds if the number of tuples in the relation having $\{x_1, x_2, \ldots x_n, y_1, y_2, \ldots, y_m\}$ is greater than the user defined minimum support (σ); also, the ratio of number of tuples having values $\{x_1, x_2, \ldots x_n, y_1, y_2, \ldots y_m\}$ to that of number of tuples having $\{x_1, x_2, \ldots x_n\}$ is greater than the user defined minimum confidence (c). Such rules are mainly based on togetherness or co-occurrence of values in a tuple. However, in the dominant entity based mining like personalized mining, we want to mine the rules of the following type : *income* > *"Range X"* $\wedge$ *age* < *"Range Y"* $\Longrightarrow$ *purchase* = *"Costly goods"*. Note that in order to generate such rules from multiple relations/databases, support for individual values are important and associations are built using the dominant entity. Such rules are called *dominant entity based association rules, (DEBAR)*. Further, in the above rule, all "Costly goods" need not be purchased together. (The goods purchased is said to be "costly" if the cost is more than "Z Dollars"). We originate a scheme for mining for dominant entity based association rules which is *not constrained to look for co-occurrence* of values in tuples. It can look at attributes in one or more databases which may be located at many places. We discuss this scheme in section 7.3.

In order to generate DEBARs, there is a need to link the values of attributes of interest (i.e., values of characteristic attributes) to the values of dominant entity attributes. We propose and develop a structure called "multi-database domain link network (MDLN)" for the mining activity that involves several databases. This structure provides link between the values of characteristic attributes and the values of dominant entity attributes in an efficient manner in terms of - *space* to hold the structure; and *access time* to discover DEBARs out of it. Here, we consider the problem of mining for associations between the set of values of attributes in the relations, R_j^i, $i = 1 \ldots n$, and $j = 1 \ldots r_i$, possibly belonging to several databases. The MDLN structure, the algorithm, the time and space requirements for its construction are discussed in section 7.4.

We show how meaningful associations can be mined using MDLN in section 7.5. We

describe the characteristics of MDLN structure with respect to DEBAR mining in section 7.6. Experimental results are described in section 7.7. We conclude our study in section 7.8.

7.3 Dominant Entity Based Association Rules (DEBAR)

Let v^x_{aij} and v^y_{bkl} be the j^{th} and l^{th} values of characteristic attributes, A^x_{ai} and A^y_{bk} which belong to relations R^x_a and R^y_b of databases D_x and D_y respectively . Let $d_1, d_2, \ldots d_N$ be the N distinct values pertaining to DEA.

Let $C(v^x_{aij})$ be the count of v^x_{aij} with respect to each d_z $(1 \leq z \leq N)$ to which it maps. This gives the *support* of v^x_{aij}. If the support of v^x_{aij} is greater than or equal to user defined minimum support (σ), then we call v^x_{aij} - a frequent value. Let the set of values d_z $(1 \leq z \leq N)$ on which count of v^x_{aij} depends be $P^{v^x_{aij}}$. This gives the set of elements which supports v^x_{aij}. This set is used to check if two frequent values satisfy the property that the number of distinct values of the dominant entity attributes to which both the frequent values are mapped is greater than or equal to user defined threshold. If so, there is a possibility for a DEBAR between these two values. Note that instead of using a single minimum threshold value σ, typically input by the user, one can use two different thresholds as follows. One threshold is to check whether the value is frequent enough to qualify as a *frequent value* (i.e., σ_1) and the other, i.e., σ_2 is used to check whether two frequent values are having enough common mappings with respect to the dominant entity attributes values. This scheme offers an additional flexibility in mining for DEBARs.

Similar to that of v^x_{aij}, let $C(v^y_{bkl})$ be the count of v^y_{bkl} with respect to each of the d_w $(1 \leq w \leq N)$ to which it maps. Let the distinct d_w $(1 \leq w \leq N)$ on which count of v^y_{bkl} depends be a set, $P^{v^y_{bkl}}$. The dominant entity based associaton rule DEBAR, $v^x_{aij} \Longrightarrow v^y_{bkl}$ holds if

1. v^x_{aij} is a frequent value,

2. v^y_{bkl} is a frequent value,

3. Number of elements in $P^{v^x_{aij}} \cap P^{v^y_{bkl}} \geq \sigma$ and

4. $|P^{v^x_{aij}} \cap P^{v^y_{bkl}}|/|P^{v^x_{aij}}| \geq c$, user defined minimum confidence, c. The L.H.S of the inequality gives the *strength* of the rule.

We have the following four cases arising out of possible variations:

1. $(a = b)$ and $(i = k)$ and $(x = y)$: i.e., associations between the values of same attribute which belongs to same relation in the same database. This leads to associations of the form $fever \Longrightarrow headache$ over the attribute *disease*.

2. $(a \neq b)$ and $(i = k)$ and $(x = y)$: i.e., associations between the values of different attributes which belong to the same relation. This leads to associations of the form (*grade* = 'senior') $\Longrightarrow$ (*salary* = 'high'), where *grade* and *salary* are two different attributes in the relation EMPLOYEE.

3. $(a \neq b)$ and $(i \neq k)$ and $(x = y)$: i.e., associations between two different attributes which belong to different relations which in turn belong to the same database. An example for this scenario is (*age* = 'range[20-30]') $\Longrightarrow$ (*project-type* = 'programming-oriented'). Here, *age* and *project-type* are two different attributes belonging to two different relations say EMP_PERMANENT_INFORMATION and PROJECT_INFORMATION.

4. $(a \neq b)$ and $(i \neq k)$ and $(x \neq y)$: i.e., associations between two different attributes which belong to relations of different databases. An example for this scenario is *salary* = *'high'* $\Longrightarrow$ (*disease* = 'heart-related disease'). Here, *salary* and *disease* are attributes belonging to different relations, say, EMPLOYEE and PATIENT-DISEASE and also different databases say COMPANY-INFORMATION-SYSTEM and PUBLIC-HEALTH-CENTRE.

We illustrate the generation of DEBARs using the following example. Consider the relations EMP_PERMANENT_INFORMATION and CUSTOMER_PURCHASE shown in

Figure 7.2 Example relations

EMP_PERMANENT_INFORMATION relation

Pid	Name	Location	Sex	Age
				30
				40
				30
				40
				30

FIGURE A

CUSTOMER_PURCHASE relation

Pid	Goods-purchased	Cost
4	bread	
4	milk	
2	coffee	
2	butter	
4	coffee	
4	butter	
4	coffee	

FIGURE B

Figure 7.2A and Figure 7.2B. Let the characteristic attributes be *Age*, *Goods-purchased* and the user specified minimum support (σ) be 2 (i.e., the number of tuples of the relation in which *a* value under consideration of an attribute occurs is atleast 2 to qualify as frequent).

In EMP_PERMANENT_INFORMATION relation, the *Age* values, 30 and 40 have enough support, i.e., 2 and 3 respectively. For *Age* value 40, the DEA mapping set, P^{40} is {2, 4}. Similarly, for *Age* value 30, the DEA mapping set, P^{30} is $\{1, 3, 6\}$.

In CUSTOMER_PURCHASE relation, $coffee$ and *butter* values of attribute *Goods-purchased* are having enough support i.e., 2. Note that the support of $coffee$ is not 3. This is because, the support is based on distinct mappings on dominant entity attribute values. Both $coffee$ and *butter* have the DEA mapping set, {2,4}. Let them be P^{coffee} and P^{butter}.

Since $P^{40} \cap P^{coffee} \geq \sigma$, $P^{40} \cap P^{butter} \geq \sigma$, $P^{coffee} \cap P^{butter} \geq \sigma$ and $P^{40} \cap P^{coffee} \cap P^{butter} \geq \sigma$, the tuples - $< 40, coffee >$, $< 40, butter >$, $< coffee, butter >$ and $< 40, coffee, butter >$ - are **frequent tuples**.

A DEBAR, $A \Longrightarrow B$ is constructed if $P^A \cap P^B / P^A \geq c$, a user defined minimum confidence. In our example with $c = 1.0$, one of the possible DEBAR is, $(Age = 40) \Longrightarrow (Goods\_purchased = coffee)$. This DEBAR shows a binary association. The DEBARs can be generated even when the values are generalized with the help of domain knowledge [Srikant et.al, 95].

Note that it is possible to generate DEBARs between **tuples** which are instantiations of Cartesian product of a collection of characteristic attributes. An example which highlights this possibility is $(Age = 45 \wedge Profession = Programmer) \Longrightarrow (Disease = Miopia \wedge Vehicle = Four\text{-}wheeler)$. This association is called an n-ary association. Here - *Age*, *Profession*, *Disease* and *Vehicle* - are the characteristic attributes which may belong to different databases. To generate the above DEBAR, the large tuple should be <45, *Programmer*, *Miopia*, *Four-wheeler*>; and $(P^{45} \cap P^{Programmer} \cap P^{Miopia} \cap P^{Four-wheeler})/(P^{Miopia} \cap P^{Four-wheeler}) \geq c$. We present an algorithm to generate DEBARs in section 7.5.

The DEBARs are different from conventional association rules (AR) [Agrawal et.al, 94] in the following aspects :

1. In AR mining, the frequency (largeness) of an itemset is based on the number of times it appears in the transaction database. It would account for every transaction made ignoring the fact that the *same* transaction is made by the *same* customer many number of times. However, in DEBAR, we are interested in mapping of values of characteristic attributes on *distinct* DEA values. For example, in the person based information system, if *many* people who have two wheelers are suffering from headache, then an interesting DEBAR could be "two-wheeler-user $\Longrightarrow$ has-headaches". Here, we do not care if the same person has many two wheelers. Because in our frame work, the frequency is based on how many DEA values are involved in the mapping process.

2. In AR mining, even if items i_a and i_b occur many number of times, and do not occur together in any transaction, then (i_a, i_b) is not a frequent itemset because of the *togetherness* property. Note that people may not purchase several costly goods *together*. Therefore, costly goods may not occur together in the same transaction of a transaction database. So, a rule like "{Airconditioner, Microwave, Refrigerator, TV} $\Longrightarrow$ Four wheeler" is not possible, because the possibility of number of transactions which have all the five items in a transaction is less likely. In the case of DEBAR, we do not insist on *together occurrence* of the values/items; rather, how

many DEA values are involved in the association process is of importance. So, the above rule is possible, if we find many persons with those goods bought, perhaps, even at different times.

3. The AR mining activity is based on a single relation. However, DEBAR mining activity can be performed on multiple relations/databases.

7.4 Multi-database Domain Link Network (MDLN)

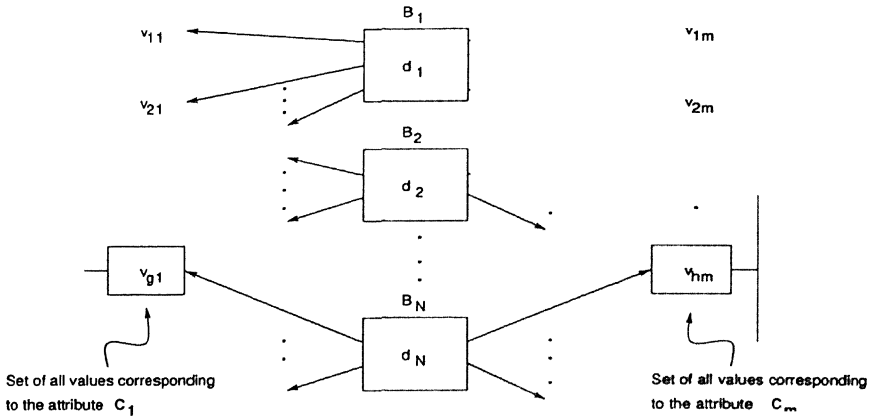
As discussed in section 7.3, we want to find the associations between the values of characteristic attributes which may belong to several databases. These values are mapped to a range of values of DEA. So, there is a need to link the values of characteristic attributes using the DEA values, with the help of which we can find the required associations. We propose a structure called multi-database domain link network (MDLN) to generate such links between the values of characteristic attributes and values of DEA across the databases.

Let $R^{i1}_{x1}, R^{i2}_{x2}, \ldots R^{il}_{xn}$, where $i, x \in \{1, \ldots l\}$, be n relations with the dominant entity attributes DEA, and belong to different databases. Let $CA^{i1}_{1x}, CA^{i1}_{2x}, \ldots CA^{i1}_{cx}$, $CA^{i2}_{1y}, CA^{i2}_{2y}, \ldots CA^{i2}_{dy}, \ldots CA^{il}_{1z}, CA^{il}_{2z}, \ldots CA^{il}_{ez}$ be m characteristic attributes (CAs) corresponding to n relations. For simplicity, let these m characteristic attributes be $\mathbb{C}_1, \mathbb{C}_2, \ldots \mathbb{C}_m$. Let the domain of DEA be $\{d_1, d_2, \ldots d_N\}$. In **person warehouse**, the values of DEA, $d_1, d_2, \ldots d_N$ map to each of <name,address> groups pertaining to N persons. [$d_1, d_2, \ldots, d_N$ could be social security numbers (SSN), if they are available]. The linking mechanism among values of m characteristic attributes (CAs) through $d_1, d_2, \ldots, d_N$ is shown in Figure 7.3. In the figure, $B_1, B_2, \ldots B_N$ are structures to hold the associations.

There are four types of relationships, **one to many**, **one to one**, **many to one** and **many to many** - that are possible between the values of characteristic attributes and DEA.

We discuss the handling of each of these relationships in the following sub-sections. In

Figure 7.3 Linking mechanism



order to realize a compact structure, we treat the relationships *one* to *many* (*one* to *one*) and *many* to *many* (*many* to *one*) between the characteristic attributes and dominant entity attributes differetly.

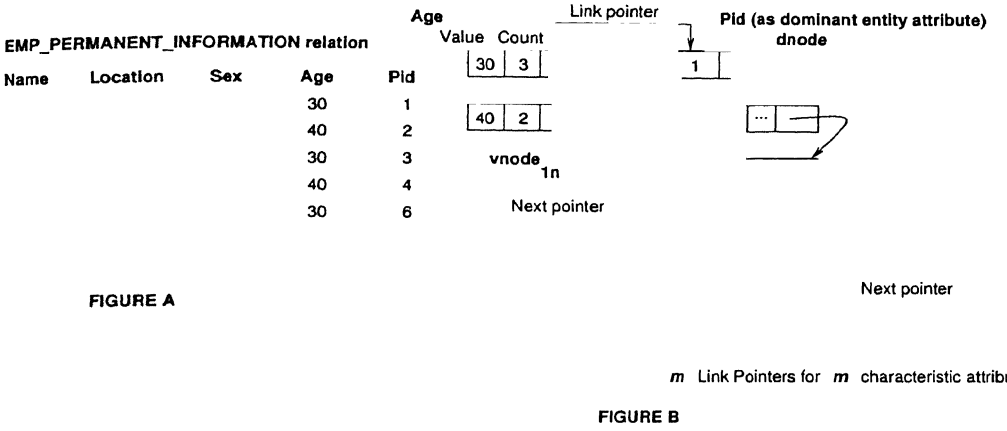
7.4.1 One to Many (1:N) relationship

The relationship between the characteristic attribute, $\mathbb{C}_i$ and the dominant entity attribute, *DEA* is $1 : N$, if each value of $\mathbb{C}_i$ maps to one or more values of the *DEA* and not vice versa.

For example, consider the relation, EMP_PERMANENT_INFORMATION shown in Figure 7.4A. Assume that the characteristic attribute is *Age* and dominant entity attribute is *Pid*. There is 1:N relationship between *Age* and *Pid*. Figure 7.4B shows the part of MDLN structure which handles the 1:N relationship. The figure shows two types of nodes - $\mathbf{vnode}_{1n}$ **(value node)** and **dnode (DEA value node)**. A $\mathbf{vnode}_{1n}$ holds a value of the characteristic attribute, and the number of times the value is mapped to distinct dominant entity values. A **dnode** holds a value of the DEA. Both $\mathbf{vnode}_{1n}$ and **dnode** also have two pointers, the *Link pointer* and the *Next pointer*.

1. *Link pointer :* This is a pointer which helps in navigation through the actual associated values of the DEA corresponding to a value of the CA in the form of a circular

Figure 7.4 Handling of 1:N relationship for m associations



list. In Figure 7.4B, from the DEA node (called **dnode**) having value 1, the link pointer points to the **dnode** having value 3 whose link pointer points to the **dnode** having the value 6. Note that these three distinct *Pid* values are associated with a value 30 of the characteristic attribute *Age*. So, a circular list is constructed involving the $\mathbf{vnode}_{1n}$ whose value = 30 and all **dnode**s which are having the associated values corresponding to *Age* = 30 using the link pointers. Since there are three *Pid* values associated with *Age* = 30, *Count* field of corresponding $\mathbf{vnode}_{1n}$ is set to 3.

Link pointer in the $\mathbf{vnode}_{1n}$ is a single pointer. However, **dnode** contains m link pointers since m characteristic attributes are involved in the relationship with the DEA.

2. *Next pointer* : This is used to generate the list of nodes pertaining to the values of a characteristic attribute or DEA.

Note that even though our representation *resembles* the structure which is used to handle 1:N relationships in the network data model, there are the following differnces between the two structures:

In the case of MDLN structure,

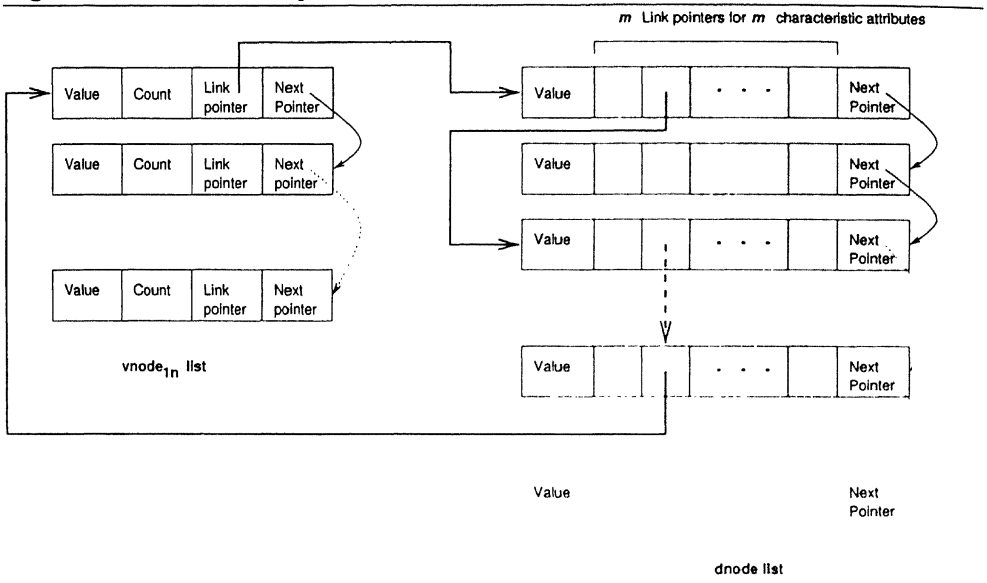
1. none of the values in the dnode list (i.e., list of dnodes) and the vnode_{1n} list (i.e., list of vnode_{1n}s) is *repeated.*

2. more than one mapping between the vnode_{1n} and the dnode is simply ignored in accordance with the definition of DEBAR.

3. each vnode_{1n} has a *Count* field for a compact realization of the structure.

An abstract representation for 1:N relationship using MDLN structure is shown in Figure 7.5

Figure 7.5 An abstract representation of 1:N relationship

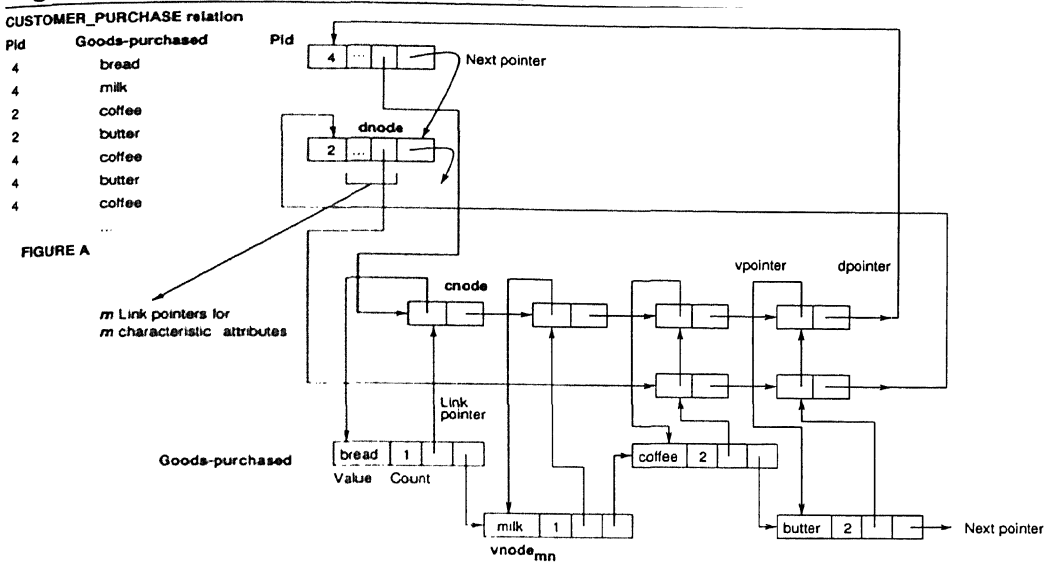


7.4.2 Many to Many (M:N) relationship

The relationship between the characteristic attribute, $\mathbb{C}_i$ and the dominant entity attribute, *DEA* is $M : N$, if values of both $\mathbb{C}_i$ and *DEA* have more than one mapping from each other.

For example, consider the relation, CUSTOMER_PURCHASE shown in Figure 7.6A. Assume that the characteristic attribute is *Goods-purchased* and dominant entity attribute is *Pid*. There is M:N relationship between *Goods-purchased* and *Pid*. Figure 7.6B shows the part of MDLN structure which handles the M:N relationship. The figure shows three types of nodes, **dnode (DEA value node)**, **vnode_{mn} (value node)** and **cnode**

Figure 7.6 Handling of M:N relationship for m associations

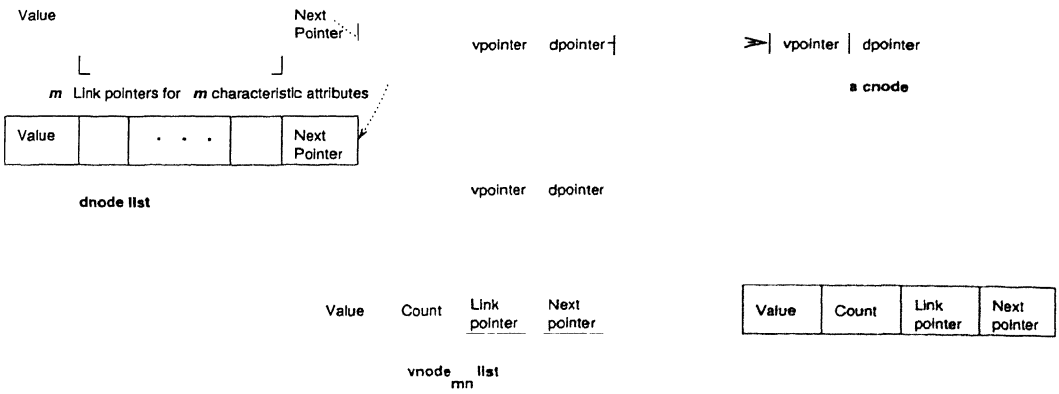


(**connector node**). Similar to $\mathbf{vnode}_{1n}$, a $\mathbf{vnode}_{mn}$ holds a value of the characteristic attribute, and the number of times the value is mapped to the distinct DEA values. A **cnode** is a dummy node which is used to visualize an M:N relationship as two 1:N relationships. There are two types of pointers in **dnode** and $\mathbf{vnode}_{mn}$: they are *Link pointer* and *Next pointer*. *Link pointer* in **dnode** points to a **cnode** and the *Next pointer* in **dnode** points to a **dnode**. But *Link pointer* of $\mathbf{vnode}_{mn}$ points to a **cnode** and *Next pointer* of $\mathbf{vnode}_{mn}$ points to a $\mathbf{vnode}_{mn}$. A **cnode** has two pointers. One is *dpointer*, pointing to a **cnode** or a **dnode**. This pointer generates a circular list involving **dnodes** and **cnodes**. Other is a *vpointer*, pointing to a **cnode** or a $\mathbf{vnode}_{mn}$. This pointer generates a circular list involving $\mathbf{vnode}_{mn}$s and **cnodes**. The *Count* field of $\mathbf{vnode}_{mn}$ has a similar functionality as that of *Count* field of $\mathbf{vnode}_{1n}$.

The differences which we quoted in section 7.4.1 also hold between our representation and the structure to handle M:N relationship in the network data model.

An abstract representation for M:N relationship using MDLN structure is shown in Figure 7.7

Figure 7.7 An abstract representation of M:N relationship



7.4.3 One to One (1:1) and Many to One (M:1) relationship

1:1 relationship is a trivial version of 1:N relationship. So, it can be handled as a specialization of the 1:N relationship. M:1 relationship can be handled as an M:N relationship which is discussed in the previous section.

Note that every value in database D is present in MDLN structure. But, it is not possible to reproduce the relevant part of D from the MDLN structure. This is because 'count' value is not maintained with respect to values stored in the **dnodes** and **cnodes** in the MDLN structure. This choice is made for two reasons : (i) It saves space and (ii) For mining DEBAR, the proposed representation is adequate. However, it is possible to make the MDLN structure a *complete* representation of D by adding

a *Count* field to **dnode**; for a 1:N relationship, this field handles multiple occurrences of a dominant entity attribute value with respect to a value of the characteristic attribute; and

a *Count* field to **cnode**; for an M:N relationship, this field handles multiple occurrences of values of dominant entity attributes as well as values of characteristic attributes.

Next, we describe the fields of nodes that are used to construct the multi-database domain link network structure.

A. DEA value node, dnode : This node is used to hold a value of the DEA and there by provide the linkage between values of characteristic attributes. For example, in Figure 7.4B and Figure 7.6B, the nodes holding *Pid* values constitute **dnodes**. We call such a list of nodes, **Dnode**. Each **dnode** has mainly three fields.

1. *Value* field : Used to hold the value of the DEA.

2. *Link pointers :* Used to provide the link to other **dnode** or **vnode** as in the case of 1:N (1:1) relationship; used to point to the **cnode** in the case of M:N (M:1) relationship. Number of link pointers in the **dnode** depend on number of characteristic attributes (CAs) involved in the association generation process.

3. *Next pointer :* Used to point to next **dnode** to maintain the list of all values of DEA.

B. CA value node, vnode ($\mathbf{vnode}_{1n}$/$\mathbf{vnode}_{mn}$) : This node is used to hold a value of the CA. For example, in Figure 7.4B and Figure 7.6B, the nodes holding *Age* and *Goods-purchased* values constitute **vnodes**; more specifically, $\mathbf{vnode}_{1n}$**s** and $\mathbf{vnode}_{mn}$**s** respectively. We call the list of **vnodes**, **Vnode**. Each **vnode** consists of following four fields :

1. *Value field* : Used to hold a value of the characteristic attribute.

2. *Count field :* It is incremented whenever the concerned **vnode** encounters a new **dnode** or a **vnode** is created for the first time. The *Value* is said to be *frequent* if its *Count* field value is greater than or equal to user defined minimum support value.

3. *Next pointer :* Used to point to next **value node** to maintain the list of all values of the characteristic attribute.

4. *Link pointer :*

(a) *Link pointer of* $\boldsymbol{vnode}_{1n}$*:* It points to a **dnode**, whose value is the first value on which $\mathbf{vnode}_{1n}$ has the association.

(b) *Link pointer of* $\boldsymbol{vnode}_{mn}$*:* It points to a **cnode**.

C. Connector node, cnode : It is used for connecting nodes between characteristic attribute values and DEA values where an M:N relationship exists. It consists of two fields. They are :

1. *dpointer :* which either points to another **cnode** or **dnode**.
2. *vpointer :* which either points to another **cnode** or $\mathbf{vnode}_{mn}$.

Different kinds of nodes : **dnode**, $\mathbf{vnode}_{1n}$ and $\mathbf{vnode}_{mn}$ are shown in Figure 7.4B and Figure 7.6B.

We give an algorithm to construct the MDLN structure along with the time and space complexity in the following sections.

7.4.4 Algorithm for Constructing the MDLN Structure

In this section, we give a detailed algorithm for the construction of MDLN.

Algorithm

INPUT :

1. Semantic network, $\mathcal{S}$.
2. A collection of n databases $D_1, D_2, \ldots D_n$. Every one of these databases can be described by a collection of nodes in $\mathcal{S}$.
3. Set of characteristic attributes, $\mathbb{C} = \{\mathbb{C}_1, \mathbb{C}_2, \ldots \mathbb{C}_n\}$.
4. Dominant entity attributes, $\mathbb{P}$.

OUTPUT : Generation of **MDLN** structure.

STEPS :

Let **Dnode** be the list of **dnode**s pertaining to the values of $\mathbb{P}$.
Let $\mathbf{Vnode}_{1n}$**[i]** be the list of $\mathbf{vnode}_{1n}$s pertaining to the values of $CA_i \in \mathbb{C}$, where CA_i is in a 1:N relationship with $\mathbb{P}$.
Let $\mathbf{Vnode}_{mn}$**[i]** be the list of $\mathbf{vnode}_{mn}$s pertaining to the values of $CA_i \in \mathbb{C}$, where CA_i is in an M:N relationship with $\mathbb{P}$.

Identify the relevant databases using $\mathcal{S}$ as discussed in module **I** of **AMAAM** described in section 4.3.

Identify the required relation in the selected set of databases each of which is having $\mathbb{C}_i$ $(1 \leq i \leq m)$ and $\mathbb{P}$.

For each $\mathbb{C}_i$ $(1 \leq i \leq m) \in \mathbb{C}$

Begin

Find the relation r with the schema R_x^y to which $\mathbb{C}_i$ belongs

For each tuple t of r

Begin

$V = r[\mathbb{C}_i]$

$P = r[\mathbb{P}]$

Case 1 : $\mathbb{C}_i$ and $\mathbb{P}$ are having 1:N (1:1) relationship :

1. Update $\mathbf{Vnode}_{1n}$**[i]** and **Dnode** lists in such a way that no repeated V and P values are stored in the respective lists. The unique V is stored in the value field of $\mathbf{vnode}_{1n}$ and the unique P value is stored in the value field of **dnode**.

2. For each $\mathbf{vnode}_{1n}$ in $\mathbf{Vnode}_{1n}$[i] construct a circular list involving itself and other nodes in the **Dnode** list which are having values in association with it. *Count* field of $\mathbf{vnode}_{1n}$ is updated in such a way that it shows the number of distinct DEA values with which it is associated.

Case 2 : $\mathbb{C}_i$ and $\mathbb{P}$ are having M:N (M:1) relationship :

1. Update$\mathbf{Vnode}_{mn}$[i] and **Dnode** lists in such a way that no repeated V and P values are stored in the respective lists. The unique V is stored in the value field of $\mathbf{vnode}_{mn}$ and the unique P is stored in the value field of **dnode**.

2. For each $\mathbf{vnode}_{mn}$ in $\mathbf{Vnode}_{mn}$[i] list, construct a circular list involving itself and **cnodes**. Similarly, for each **dnode** in **Dnode** list, construct a circular list involving itself and **cnodes**. Both circular lists are constructed in such a way that, a **cnode** which is the part of both the circular lists indicates the association of the value in a $\mathbf{vnode}_{mn}$ in one circular list with the value of a **dnode** in the other circular list. *Count* field of $\mathbf{vnode}_{mn}$ is updated in such a way that it shows the number of distinct DEA values with which it is associated.

End

End

7.4.4.1 Space and Time Requirements to Construct the MDLN Structure :

Space requirement :

Size of each **dnode** is given by $S_{Value} + S_{Linkpointer} \times m + S_{Nextpointer}$. Here, S_{Value} is the size of Value field, $S_{Linkpointer}$ and $S_{Nextpointer}$ are the sizes of Link pointer and Next pointer respectively, and m is the number of characteristic attributes. So, total size of N

distinct **dnodes** is

$$S_{Dnode} = (S_{Value} + S_{Linkpointer} \times m + S_{Nextpointer}) \times N.$$

Size of each $\mathbf{Vnode_{1n}[i]}$ list is given by $(S_{Value} + S_{Linkpointer} + S_{Count} + S_{Nextpointer}) \times |d(CA_i)|$. Here, S_{Count} is the size of Count field, $|d(CA_i)|$ is the size of domain of characteristic attribute, CA_i (or distinct sub-domain of CA_i, if associations are among the values of CA_i itself). Let on an average, $m/2$ be the number of characteristic attributes which have 1:N relationship with the DEA. In that case, total size of $\mathbf{Vnode_{1n}}$ is

$$S_{Vnode_{1n}} = \sum_{i=1}^{m/2}((S_{Value} + S_{Linkpointer} + S_{Count} + S_{Nextpointer}) \times |d(CA_i)|)$$

Similarly, for the remaining $m/2$ characteristic attributes which have M:N relationship with the DEA, the total size of $\mathbf{Vnode_{mn}}$ is given by

$$S_{Vnode_{mn}} = \sum_{i=m/2}^{m}((S_{Value} + S_{Linkpointer} + S_{Count} + S_{Nextpointer}) \times |d(CA_i)|) +$$

$$\sum_{i=m/2}^{m}((S_{dpointer} + S_{vpointer}) \times N \times |d(CA_i)|)$$

The second term in the above expression is the total size of all **cnodes**.

So, total size of the MDLN structure is

$$S_{MDLN} = S_{Dnode} + S_{Vnode_{1n}} + S_{Vnode_{mn}} \tag{7.1}$$

In order notation, the expression(7.1) reduces to

$\bigcirc(m \cdot N \cdot L)$ where, $L = |d(CA)|$, assuming that $|d(CA_i)| = |d(CA)|$ for $i = 1 \ldots m$.

Time requirement :

Time required to construct the MDLN structures is given by the product of the total number of disk accesses and the access time, where access time is constant. The total number of disk accesses is obtained as follows.

Let $R^{i1}_{x1}, R^{i2}_{x2}, \ldots, R^{il}_{xn}$, where $ix \in \{1, \ldots l\}$, be n relations and the associations among the values of their characteristic attributes are to be mined. Let $b^{i1}_{x1}, b^{i2}_{x2}, \ldots, b^{il}_{xn}$ be the number of disk blocks for each of the relations. To build the MDLN structure, we need to access each relation *only once*. So, to construct the MDLN structure, the number of disk accesses required is

$$b^{i1}_{x1} + b^{i2}_{x2} + \cdots + b^{il}_{xn} \tag{7.2}$$

In order notation, the expression (7.2) reduces to

$$\bigcirc(n \cdot b)$$

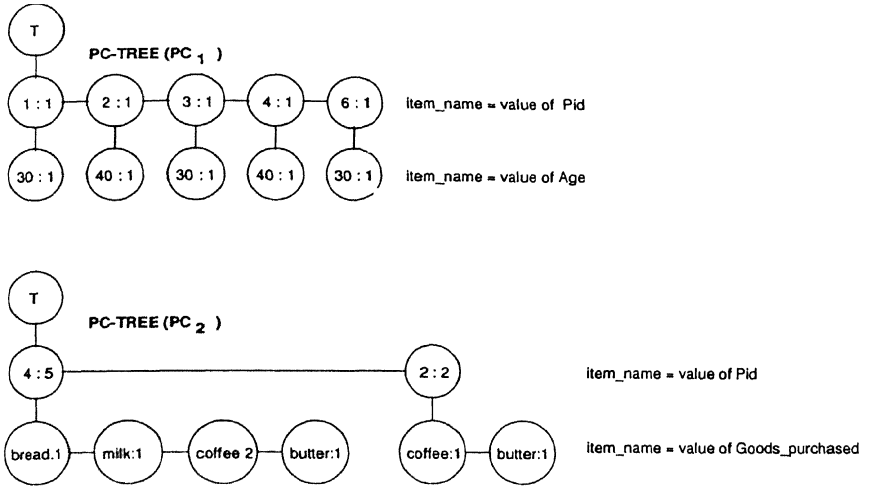
where, b is the average number of disk blocks for a relation.

We already established in chapter 6 that PC-tree based scheme for mining multiple databases generates smaller structure when compared with EIF based scheme. And also, navigation across a collection of PC-trees (each corresponds to a database) can be done using the DEA values as in the MDLN structure. So, we compare the space requirements of MDLN structure with PC-tree based structure for multiple databases.

Size of each PC-tree node is given by $S_{item\text{-}name} + S_{Count} + 2 \times S_{LinkPointer}$. Here, $S_{item-name}$ is the size of 'item-name' field, S_{Count} is the size of "count" and $S_{LinkPointer}$ is the size of the pointer which is "child" or "sibling". For the relations shown in Figure 7.4A and Figure 7.6A, the PC_1 and PC_2 in Figure 7.8 are the corresponding PC-trees where the attributes under consideration for PC_1 are *Pid* and *Age*; and for PC_2, the attributes under consideration are *Pid* and *Goods-purchased*.

Since there are n relations under consideration, n PC-trees are constructed. On an average let $x = m/n$ $(m \geq n)$ be the number of characteristic attributes distributed

Figure 7.8 PC-trees for the relations shown in Figure 7.4A and Figure 7.6A



across n relations. Each characteristic attribute may have either 1:N relationship or M:N relationship with the DEA. If all x attributes of a relation i are having 1:N relationship with the DEA, then the size of the PC-tree for N distinct DEA values is given by

$$S_{PC\text{-}tree^i} = (S_{item\text{-}name} + S_{Count} + 2 \times S_{LinkPointer}) \times (N + N \times x)$$

If all m attributes in the relations have 1:N relationship with the DEA, then the total size of n PC-trees is

$$S_{PC\text{-}tree^{1n}} = \sum_{i=1}^{n} (S_{PC\text{-}tree^i})$$

In order notation, this is equal to

$$\bigcirc(n \cdot N \cdot m) \tag{7.3}$$

The total MDLN structure size if all m attributes have 1:N relationship with the DEA is

$$= \bigcirc(L \cdot m) \text{ where, } L = |d(CA)|, \text{ assuming that } |d(CA_i)| = |d(CA)| \text{ for } i = 1 \ldots m. \tag{7.4}$$

So, in this case, the space requirement for the MDLN structure is smaller than that of the PC-tree based representation.

On the other hand if all x attributes of a relation j are having M:N relationship with the DEA, then the size of the PC-tree for N distinct DEA values is given by

$$S_{PC-tree^j} = (S_{item-name} + S_{Count} + 2 \times S_{LinkPointer}) \times$$

$$(N + N \times |d(CA_1)|) + \cdots + (N \times |d(CA_1)| \times \cdots |d(CA_x)|)$$

If all m attributes in the relations have M:N relationship with the DEA, then the total size of n PC-trees is

$$S_{PC-tree^{mn}} = \sum_{j=1} (S_{PC-tree^j})$$

In order notation, this is equal to

$$\bigcirc(n \cdot N \cdot L^m) \text{ where, } L = |d(CA)|, \text{ assuming that } |d(CA_i)| = |d(CA)| \text{ for } i = 1 \ldots m. \tag{7.5}$$

The total MDLN structure size if all m attributes have M:N relationship with the DEA is

$$\bigcirc(m \cdot N \cdot L) \text{ where, } L = |d(CA)|, \text{ assuming that } |d(CA_i)| = |d(CA)| \text{ for } i = 1 \ldots m. \tag{7.6}$$

So, the space requirement for the MDLN structure is smaller than that of the PC-tree based representation, in this case also.

From (7.3), (7.4) and (7.5), (7.6) - it is clear that space requirements for MDLN structure is smaller than that of the PC-tree structure.

As in MDLN structure, to build the PC-tree, we need to access each relation *only*

once.

For illustration purpose, consider the relations shown in Figure 7.4A and Figure 7.6A each with one characteristic attribute, *Age* and *Goods-purchased* respectively. The MDLN structure is the combination of Figure 7.4B and Figure 7.6B on DEA values. Let the storage space of each field of a node be 2 Bytes. In that case, the total storage requirement for MDLN structure is $2 \times 8 + 5 \times 8 + 4 \times 8 + 6 \times 4 = 112$ Bytes.

Corresponding to the relations given in Figure 7.4A and Figure 7.6A, PC-trees are shown in Figure 7.8. From the figure it is clear that that total PC-tree size requirement is $20 \times 8 = 160$ Bytes.

7.5 DEBAR Generation Using MDLN Structure

From the definition of DEBARs (refer section 7.3), it is clear that we are interested in the associations between the values which are mapped to a range of values of DEA such that the cardinality of the range is greater than or equal to the user defined threshold value. Note that repeated mappings between the value of a characteristic attribute and the value of DEA are ignored.

In MDLN structure, we are not storing any information necessary to get the *togetherness* /co-occurrences of values in a tuple. And also the *Count* field of the node pertaining to a value of the CA is updated *only* if it maps to different DEA values. For example, in Figure 7.6B even though the relation has three instances of $coffee$, the *Count* value of the node is set to 2 indicating that two distinct *Pids* are having association with $coffee$.

We give a general outline of the DEBAR generating algorithm using the MDLN structure below.

Algorithm

INPUT :

1. Two sets of characteristic attributes $\mathbb{A}$ $(= \{A_1, A_2, \ldots A_n\})$ and $\mathbb{B}$ $(= \{B_1, B_2, \ldots B_m\})$, where $\mathbb{A}, \mathbb{B} \subset \mathbb{C}$.

2. MDLN structure.

3. User defined minimum support, σ and confidence, c.

OUTPUT : All possible DEBARs between tuples which are instantiations of Cartesian product of $\mathbb{A}$ and $\mathbb{B}$.

STEPS :

1. For each tuple, V_A, $< v_{A1}, v_{A2}, \ldots v_{An} >$ - an *n-tuple* which is the instantiation of Cartesian product of $\mathbb{A}$, find the frequency as follows :

   (a) Find the frequency of v_{Ax}, i.e., Count(v_{Ax}) ($1 \leq x \leq n$) using MDLN structure.

   (b) $\forall v_{Ax}$, if Count(v_{Ax}) $\geq \sigma$, then find $P^{V_A} = P^{v_{A1}} \cap P^{v_{A2}} \cap \ldots P^{v_{An}}$. If cardinality of $P^{V_A} \geq \sigma$, then V_A is frequent (large) tuple.

2. For each tuple, V_B, $< v_{B1}, v_{B2}, \ldots v_{Bm} >$ - an *m-tuple* which is the instantiation of Cartesian product of $\mathbb{B}$, find the frequency as follows :

   (a) Find the frequency of v_{Bx}, i.e., Count(v_{Bx}) ($1 \leq x \leq m$) using MDLN structure.

   (b) $\forall v_{Bx}$, if Count(v_{Bx}) $\geq \sigma$, then find $P^{V_B} = P^{v_{B1}} \cap P^{v_{B2}} \cap \ldots P^{v_{Bm}}$. If cardinality of $P^{V_B} \geq \sigma$, then V_B is frequent (large) tuple.

3. For any two frequent tuple, V_A and V_B If $(P^{V_A} \cap P^{V_B}) \geq \sigma$, then $< V_A, V_B >$ is a frequent tuple.

4. For any frequent tuple, $< V_A, V_B >$

   - if $\frac{P^{V_A} \cap P^{V_B}}{P^{V_A}} \geq c$, then output the DEBAR, $V_A \Longrightarrow V_B$.
   - if $\frac{P^{V_A} \cap P^{V_B}}{P^{V_B}} \geq c$, then output the DEBAR, $V_B \Longrightarrow V_A$.

Let S_{MDLN} be the size of MDLN structure and B be the disk block size. The number of disk access required to generate DEBARs is S_{MDLN}/B.

At any point of time, the list of **dnodes** (storage space - S_{Dnode} as discussed in section 7.4.4.1) should be there in the main memory while generating the associations. For example, corresponding to the MDLN structure shown in Figure 7.4B and Figure 7.6B, only **5 dnodes** should be memory resident at any point of time.

To generate the associations using PC-trees, the number of disk accesses required is $S_{PC-tree}/B$. Here, $S_{PC-tree}$ is the sum of sizes of all the PC-trees involved in the association process. At any point of time, it is not adequate if a PC-tree or a part of the PC-tree is made memory resident during the entire associations generation process as in the case of MDLN structure. So, if the structure (MDLN / all PC-trees) does not fit into the main memory, the data which moves in and out of main memory is less in the case of MDLN structure than that of the PC-tree structure.

7.6 Characteristics of MDLN Structure with respect to DEBAR Mining

Definition 7.6.1 Value-complete representation : A representation R is value-complete with respect to the database D, if every value of every relevant attribute of D is present in R.

Lemma 7.6.1: MDLN structure is a value-complete representation of D

Proof: Let D be $\{D_1, D_2, \ldots D_k, \ldots D_n\}$, a set of databases considered for DEBAR mining. Let A_{ij}^k be the attribute belonging to database D_k. Let the domain of A_{ij}^k be $\{v_{ij1}^k, v_{ij2}^k, \ldots v_{ijd}^k\}$ with d distinct values. If A_{ij}^k is a CA (or it belongs to DEA), then MDLN structure should have d **vnodes** (or **dnodes**) with values $\{v_{ij1}^k, v_{ij2}^k, \ldots v_{ijd}^k\}$. Since *all values* in the domain of A_{ij}^k are present in MDLN structure, MDLN structure is value-complete with respect to D_k. The above explanation holds good for any relevant attribute belonging to any D_i in D. So, MDLN structure is

value-complete with respect to D.

Definition 7.6.2 Compact representation : A representation R is a compact representation of D, if R is value-complete with respect to D and $R_s < D_s$, where R_s is the size of R and D_s is the size of D.

Lemma 7.6.2: MDLN structure is a compact representation of D.

Proof: MDLN is a compact representation because of the following two factors:

1. We ignore non-interesting attributes of the database.
2. Each value of an attribute under consideration appears *only once* in the MDLN. Multiple occurrences of values are handled by properly maintaining counts.

It is shown in chapter 6 that PC-tree size is smaller than that of the database size. We showed in section 7.4.4.1 that the MDLN structure is more compact than that of the PC-tree. So, MDLN is a compact representation with respect to D.

Definition 7.6.3 Module : A **Vnode** is called a module.

Definition 7.6.4 DEBAR-Modular : A structure, that represents values of all characteristic attributes is DEBAR-modular, if no two modules are needed to be present in the main memory at the same time for mining.

Lemma 7.6.3: MDLN is a DEBAR-modular structure.

Proof: In MDLN structure, each **Vnode** ($\mathbf{Vnode}_{1n}$ / $\mathbf{Vnode}_{mn}$) list can be considered along with the **Dnode** list for mining. For DEBAR mining, we can have **Dnode** list as memory resident and each of the modules, i.e., **Vnode** lists can be brought in any order as and when required.

This modularity property also plays an important role in addition, deletion and modification of values in the MDLN structure without bringing the whole MDLN

structure into the main memory. This helps in efficient incremental updation of the structure.

Lemma 7.6.4: Ordering of attributes is unimportant while using a DEBAR-modular structure.

Proof: Due to the modularity, any **Vnode** list can be brought in to the main memory for mining. So, ordering of attributes is not required to find associations.

Definition 7.6.5 An appropriate representation for incremental mining of DEBAR : A representation R is appropriate for incremental mining of DEBAR if it satisfies the following properties :

1. R is a value-complete and a compact representation of D.
2. R can be directly used to handle changes in D without scanning D to generate DEBAR.

Lemma 7.6.5: MDLN is an appropriate representation for incremental mining of DEBARs.

Proof: MDLN satisfies property 1 of the definition 7.6.5 from Lemma 7.6.1 and Lemma 7.6.2. MDLN structure can also handle the changes in D to $\acute{D}$ due to efficient deletion / addition / modification without revisiting D. This satisfies property 2 of definition 7.6.4.

Lemma 7.6.6: DEBAR generation using MDLN structure is independent of the order of transactions.

Proof: Follows from the construction of the MDLN structure; where MDLN structure is not constrained to look for co-occurrence of values in tuples.

Other characteristics of the MDLN structure are :

1. MDLN has the flavor of the network model, but different from the network model because

- In the network model, there is a node for each value of the attribute. However, in MDLN for the same discrete values, we have a single node pertaining to the attribute. The count field associated with the node is properly updated to show the repetition of discrete values.

- MDLN structure is built based on several databases.

2. MDLN handles

   - associations between values of a single characteristic attribute of a relation

   - associations between values of different characteristic attributes of a relation or between the relations of the same or different databases.

   - n-ary associations among the characteristic attributes.

7.7 Experimental Results

We implemented MDLN structure to handle both 1:N (1:1) and M:N (M:1) relationships between DEA and the characteristic attributes. We conducted experiments using the MDLN structure to find the associations between values based on simulated data. We also conducted experiments to study the storage requirement patterns of the MDLN structure. It is already shown in chapter 6 that the size of PC-tree structure is less than that of the database. Here, we examine whether the MDLN structure is more compact than the PC-tree or not. And also examine the number of disk accesses to generate the association rules by using both the structures. For this, we assumed the disk block size = $2K$ Bytes and used three parameters : *number of tuples*, *domain size of DEA* and *number of attributes (linkages)*. In order to make the PC-tree attribute-order independent, we constructed PC-trees each with values of DEA and values of characteristic attribute. So, <value of DEA, value of CA> constitute a pattern based on which PC-tree is constructed. In order to handle multiple databases, there is a need to link across the PC-trees through the DEA values. So, the parameter *number of attributes (linkages)* typically indicates

number of PC-trees constructed and in the case of MDLN structure, it represents number of linkages. We have two cases.

Case 1 : One to Many relationship between Characteristic and Dominant entity attributes :

Figure 7.9 shows the size of the MDLN structure with the corresponding PC-tree size. Here, the *number of tuples* is varied while size of the *domain of DEA* is fixed at 999 and *number of attributes (linkages)* is changed from 5 to 10. It is clear from both the graphs that there is no further increase in the sizes of the structures with increase in the number of tuples.

Figure 7.9 MDLN structure size Vs. PC-tree size, where the size of domain of DEA = 999

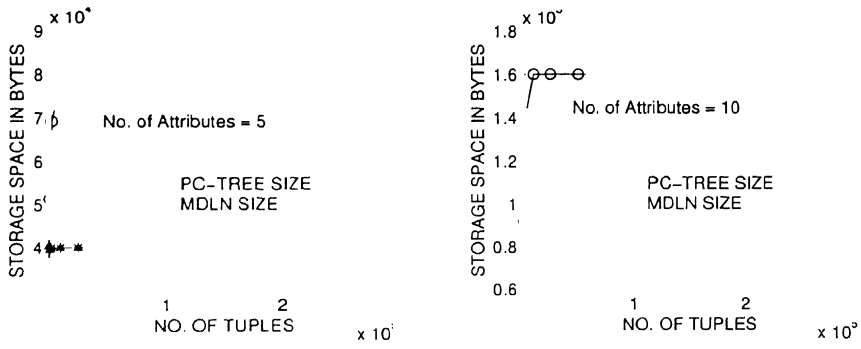


Figure 7.10 shows the sizes of MDLN structure and the corresponding PC-tree where the *number of tuples* is varied; the size of the *domain of DEA* is 9999 and and *number of attributes (linkages)* is changed from 5 to 10.

It is clear from Figure 7.9 and Figure 7.10 that since MDLN structure depends a) on the size of the domain of the attributes and not on the number of tuples, and b) for each value of an attribute only one node is created, the size does not increase as that of PC-tree and the storage requirement is about 50% of that of the PC-tree.

Figure 7.11 shows the increase in the number of disk accesses as the number of attributes changes from 5 to 10 while generating DEBAR. The graph shows that number of disk accesses required for generating DEBAR using MDLN structure is

Figure 7.10 MDLN structure size Vs. PC-tree size, where the size of domain of DEA entity = 9999

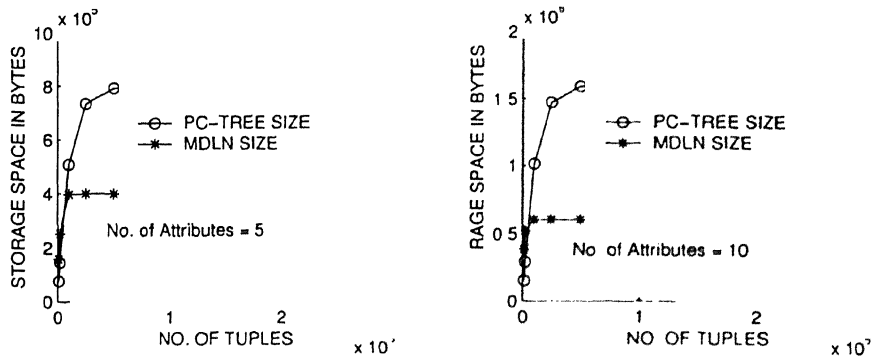
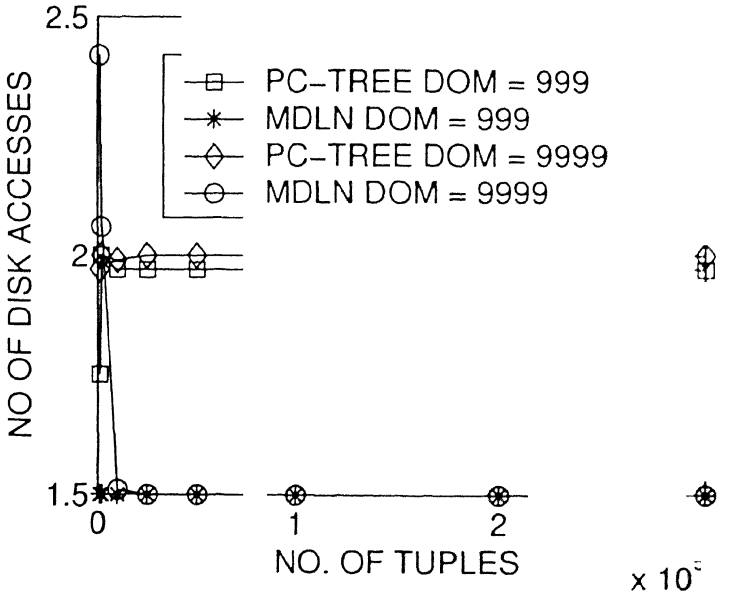


Figure 7.11 Increase in number of disk accesses

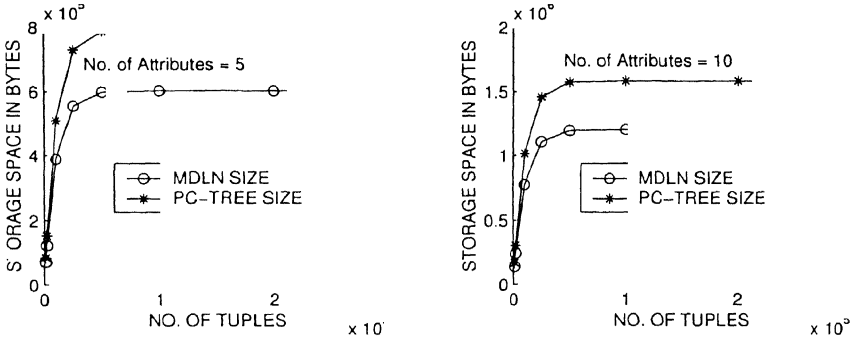


about 50% of that of the PC-tree. The graph also shows that in the case of MDLN structure, as the number of tuples increases, there is no increase in the disk accesses as the number of attributes changes from 5 to 10 with respect to the dominant entity attribute domain sizes varying from 999 to 9999. This is possible because the entire structure fits into the main memory. However, in case where it does not fit into the main memory, the DEBAR-modularity of the MDLN structure can be exploited further.

Case 2: Many to Many relationship between Characteristic and Dominant entity attributes :

In **Case 1**, the characteristic attribute values are in 1:N relationship with DEA. In **Case 2**, the characteristic attribute values have M:N relationship with the DEA. Similar to **Case 1**, two sets of experiments are conducted in this case also. Figure 7.12 shows the sizes of MDLN structure and the corresponding PC-tree where the *number of tuples* is varied while the size of the *domain of DEA* is fixed at 99 and *number of attributes (linkages)* is changed from 5 to 10. Here, we chose a smaller

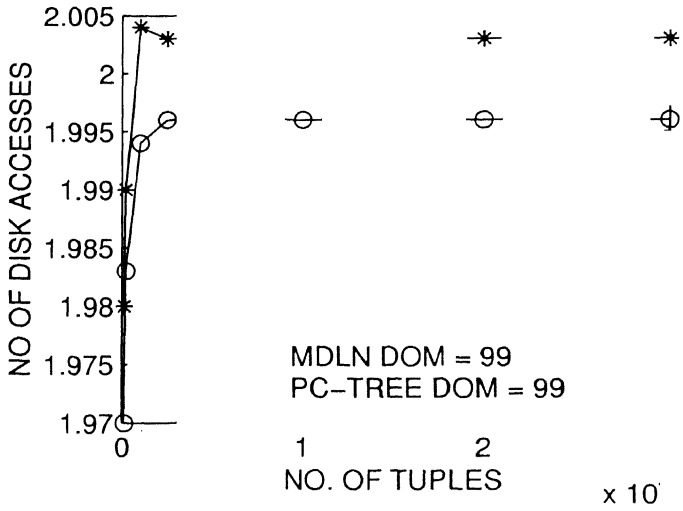
Figure 7.12 MDLN structure size Vs. PC-tree size, where the size of domain of DEA = 99



domain to show that as in 1:N relationships, there is no increase in sizes of the structures as the number of tuples increases. In this case, storage requirement for MDLN structure is about 75% of that of the PC-tree.

Figure 7.13 shows the increase in the number of disk accesses as the number of

Figure 7.13 Increase in number of disk accesses



attributes changes from 5 to 10 while generating DEBAR. The graph shows that number of disk accesses is about 90% of that of the PC-tree.

Table 7.1 MDLN structure size Vs. PC-tree size, where the size of domain of DEA = 999

No. of tuples	No. of attributes (linkages) = 5		No. of attributes (linkages) = 10	
	PC-tree Size	MDLN Size	PC-tree Size	MDLN Size
1000	130848	163228	261680	306872
2000	228592	245704	457472	471780
10,000	875792	736704	1731904	1453660
25,000	2055104	1621188	4110208	3222239
50,000	3981456	3065952	7963840	6112620
1,00,000	7691184	5848248	15385680	11679000
2,00,000	14579600	11014560	29162816	22011852
3,00,000	20809968	15687336	41616272	31351944
Normalized size ($\approx$)	1	0.75	1	0.75

Table 7.1 shows the MDLN structure size with corresponding PC-tree size where *domain of DEA* is fixed at 999 and the *number of attributes (linkages)* is varied from 5 to 10. This table shows that with the domain size of DEA = 999, MDLN structure size is about 75% of that of PC-tree size. This table also shows that as

the number of tuples increases, there is less increase in the sizes of MDLN structure and PC-tree structure hinting that at some point, further increase in the number of tuples will not alter the sizes of MDLN structure and PC-tree structure.

Table 7.2 MDLN structure size Vs. PC-tree size, where the size of domain of DEA = 9999

No. of tuples	No. of attributes (linkages) = 5		No. of attributes (linkages) = 10	
	PC-tree Size	MDLN Size	PC-tree Size	MDLN Size
1000	156304	312800	312864	688340
2000	305424	556640	610880	1122700
10,000	1307168	1631012	2614000	3067220
25,000	2733248	2816532	5467632	5434528
50,000	4793632	4392452	9586960	8484464
1,00,000	8795888	7396768	17592096	14593788
Normalized size ($\approx$)	1	0.84	1	0.83

The same behaviour can also be observed in Table 7.2. Table 7.2 shows the MDLN structure size with corresponding PC-tree size where *domain of DEA* is fixed at 9999 and the *number of attributes (linkages)* is varied from 5 to 10. From Table 7.2 it is clear that storage space requirement for MDLN structure is about 84% of that of the PC-tree.

From the experimental results it is clear that MDLN structure handles 1:N relationship between CA and DEA in a more compact manner than M:N relationship between CA and DEA.

7.8 Summary

We introduced, in this chapter, the notion of mining multiple databases for Dominant Entity Based Association Rules (DEBARs). These rules -

are dominant entity attributes oriented,

do not need explicit co-occurrences of values in tuples,

capture n-ary relationships and

can be generated from values of characteristic attributes across several relations/databases.

We showed the importance of such a mining activity by taking a practical example like personalized mining. We proposed a novel structure called Multi-database Domain Link Network (MDLN) which can be used to explore associations between values of various attributes which belong to same or different relations which in turn belong to the same or different databases. It is shown that *only one* scan of each of the identified databases is required to generate the MDLN structure. We gave a concrete framework for mining for DEBARs in single and multiple relations belonging to one or more databases using the MDLN structure.

Our experiments reveal that the storage requirement for MDLN structure varies from 50% to 84% of that of the PC-tree based representation; and the access time to generate DEBARs is about 50% to 90% of that of the PC-tree. Further, the DEBAR-modular property of the MDLN structure permits it to be used for incremental mining with very large size databases.

So, MDLN structure is the most appropriate data structure for representing multiple databases for mining associations.

Chapter 8

Discussion and Conclusions

In this chapter, we compare the schemes proposed in the thesis with some of the well-known schemes for mining association rules. We present a comparative study of schemes in section 8.1. In this section, we compare the schemes based on following four issues : (i) in section 8.1.1, we discuss the structures generated and used by various schemes, (ii) in section 8.1.2, we present knowledge based schemes, (iii) we discuss multiple database based schemes in section 8.1.3 and (iv) we summarize single database based schemes in section 8.1.4. We use one or more existing appropriate algorithms based on their applicability along with our contributions for comparison based on each of the issues mentioned above. We summarize our research contributions and point out directions for future work in section 8.2.

8.1 Comparative Study

8.1.1 Structure Based Schemes

One of the recent and efficient schemes for mining a transaction database for association rules is based on a tree structure called the Frequent Pattern Tree (FP-tree) [Han et.al, 00]. The superiority of FP-tree based scheme for ARM over well-known ARM algorithms is already established by [Han et.al, 00]. So, we compare our schemes based on Extended

Inverted File (EIF) [Chapter 4], Pattern Count-tree (PC-tree) [Chapter 6] and Multi-database Domain Link Network (MDLN)[Chapter 7] with the scheme based on FP-tree. The requirements of the schemes which generate and use a structure are - *number of database scans to generate structure*, *number of database scans required to discover frequent set of values* using the structure and *compactness* of the structure.

We discuss the above requirements under following two cases :

Case 1 : Input is a transaction database : In order to compare our structures with the FP-tree, we have chosen transaction databases [Quest site] having number of tuples 5K, 10K and 100K. Each database consists of data corresponding to attributes - *customer-id*, *items-purchased* and *customer-age*. For MDLN structure, *customer-id* is used as dominant entity, *items-purchased* and *customer-age* are characteristic attributes; *items-purchased* has M:N relationship with *customer-id* and *customer-age* has N:1 relationship with *customer-id*. For PC-tree and FP-tree, the respective structure is built using *items-purchased* and *customer-age*. For building EIF, we have chosen *items-purchased* and *customer-age* as the two attributes between the values of which we have to find associations. We considered that there are seven generalized values for attribute *customer-age*. Table 8.1 shows the size of the PC-tree, FP-tree, MDLN and the EIF structure and also number of database scans required for their construction. From Table 8.1, it is clear that size of FP-tree is

Table 8.1 Comparison : Structure sizes in bytes and number of database scans

	FP-tree	PC-tree	MDLN	EIF
Data set-5K	333776	389616	934132	12546
Data set-10K	614176	719104	1592520	2858975
Data set-100K	9088144	11140720	22671108	30481153
Number of database scans	2	1	1	

the smallest among the structures considered and its size is 25% of that of EIF, 38% of that of MDLN and 84% of that of the PC-tree size on an average. But, FP-tree construction requires *two* database scans, whereas PC-tree and MDLN-structure can be constructed using *one* database scan. Construction of EIF structure requires *seven* database scans which depends on number of generalized values for

the attribute *customer-age*. Even though the size of FP-tree is smaller than that of the PC-tree, we showed in chapter 6 that FP-tree can be constructed faster from PC-tree than its direct construction from the database. And also, FP-tree is not a *complete* representation of the database. As a consequence, it cannot be used for incremental/dynamic mining.

Case 2 : Input is multiple databases Use of FP-tree for finding associations in multiple databases is not reported in the literature. So, we will not consider it here. From Table 8.1, it is clear that EIF is as not compact as the PC-tree or the MDLN structure and also the number of database scans depends on the number of generalized values of the attribute under consideration in each of the relations of the database. So, we will not consider it also for further discussion. In chapter 7, we have shown experimentally that with the increase in number of attributes, MDLN structure is smaller in size by 50% to 84% of that of the PC-tree. It is also shown in chapter 7 that the number of scans of each of the databases considered for generating MDLN structure is *one*. This is because, the valuetable is generated implicitly and MDLN structure is constructed directly from selected relations of multiple databases. In constructing PC-tree, there is a need to scan the valuetable only once. However, explicit generation of valuetable, which is required as input for constructing PC-tree may need multiple scans of databases.

Observation : *From the above two cases, it is clear that MDLN structure is the most compact, and also can be constructed using a single scan of each of the database involved. FP-tree is the most compact structure, if the input is a transaction database and it can be constructed faster using the scheme proposed in section 6.2.2 of the thesis.*

8.1.2 Knowledge-Based Schemes

Data driven mining algorithms work directly on the data in the database. Since number of frequent tuples generated by these algorithms is typically large, many of them may not be interesting to the user. So, one of the alternatives to make association rules more interesting is by using domain knowledge for mining.

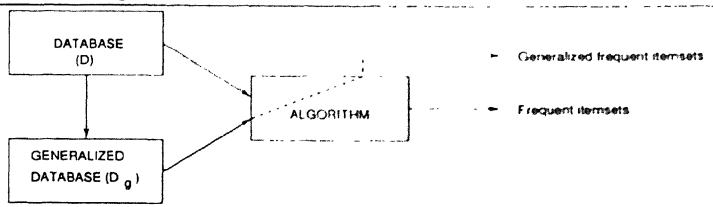
Algorithms reported in [Savasere et al., 98; Tsur et al., 98], use an "is_a" hierarchy to represent the domain knowledge. We discussed a more promising generalization in chapter 5 based on an And-Or (AO) taxonomy. Generalization of PC (GPC) and LOFP (GLOFP) trees is discussed in chapter 6.

We examine some of the implications of using knowledge for mining associations. Three important parameters that affect the performance of a mining algorithm are : the nature of knowledge, number of database scans and number of candidate tuples generated and tested. The most frequently used form of knowledge for association rule mining is *is_a* taxonomy [Srikant et.al, 95]. In the case of *is_a* taxonomy, typically, a lower level concept is linked with the higher level concept through *is_a* relation. We have used a richer version of knowledge representation structure in the form of AO taxonomy. In addition to *is_a* relation, we used *is_part_of* relation to link two concepts in the AO taxonomy. The most general form of knowledge representation structure is the semantic network. Semantic network can employ other relations in addition to *is_a* and *is_part_of*. So, an *is_a* taxonomy or an AO taxonomy is a semantic network, but not the other way. There are two kinds of mining algorithms, reported in literature, based on the type of knowledge used. These are :

Algorithms for mining generalized values

Here a database D is generalized to D_g, with the help of a multi-level "is_a" taxonomy. Typically, the same mining algorithm can be used on either the original database or the generalized database as shown in Figure 8.1. Further, generalized

Figure 8.1 Generation of generalized itemsets



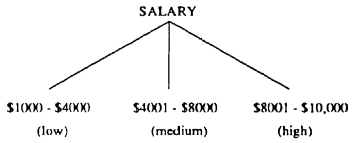
values are generated from generalized databases. For example, the data-driven algorithms like Apriori, FP-tree based algorithm, PC-tree based algorithm are used

to generate generalized itemsets. An important property of generalization is given below :

Property : Generalization does not reduce the number of database scans but it may reduce the number of candidates.

Generalization changes D to D_g, where $|D| = |D_g|$. So, the number of database scans does not change. However, a value in D_g can have a larger support because each value of an attribute in D_g is made up of a range of values of the same attribute in D. For example, consider Figure 8.2. Here, tuples having different salary values

Figure 8.2 Is_a hierarchy



in the range ($1000...$10,000) are mapped to generalized tuples having values 'low', 'medium', 'high'. So, the number of candidate itemsets might be reduced.

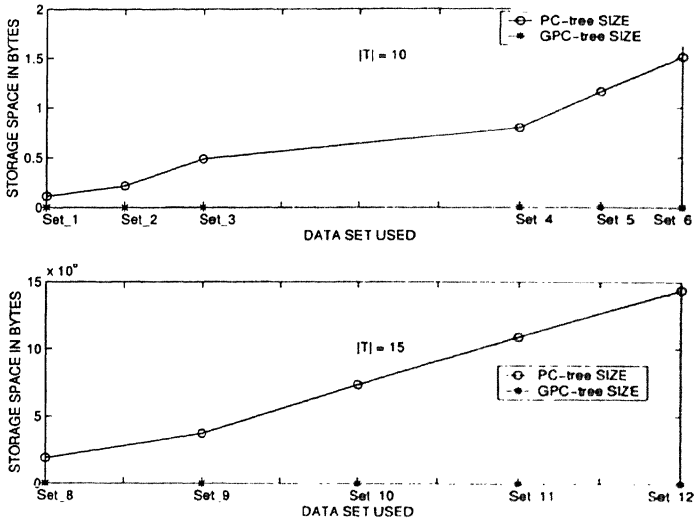
To examine this issue with a large database, we conducted the following experiment.

This experiment is conducted using algorithms to generate PC-tree and GPC-tree using the data sets Set_1 to Set_12 discussed in chapter 6. We compare the memory requirements of PC-tree and the corresponding generalized PC-tree (GPC-tree). Generalization of items is done by mapping 1000 items to 10 generalized values uniformly. The memory requirements to hold PC-tree and GPC-tree are shown in Figure 8.3. Both graphs show the normalized memory size for PC-tree and GPC-tree w.r.t GPC-tree. Due to generalization, there is a reduction in the number of candidate itemsets by a factor of 1644 to 21466. This observation is supported by Figure 8.3.

Knowledge oriented mining

Here, knowledge is used for mining only those itemsets/concepts that are of interest to the user. Algorithms based on AO taxonomy [Chapter 5] belong to this category.

Figure 8.3 Generalization Vs. non-generalization



The algorithm that exploits knowledge in the best possible way is the AO taxonomy based algorithm. It permits the usage of high level functional descriptions in addition to the 'is_a' links. As a consequence, it offers a greater flexibility in describing associations. Also, it helps in the reduction of number of candidates itemsets generated and number of database scans required compared to the data-driven algorithms. The experimental results discussed below justify this claim.

We conducted the experiment using AO-Single scan algorithm and other candidate based data-driven algorithms like Apriori and Partition based algorithm (after merging phase) using Set_3 (discussed in chapter 6) as the data set. Table 8.2 shows the number of candidate tuples (itemsets) and number of frequent tuples (itemsets) generate and tested by AO-Single scan and other data-driven algorithms for the same set of data (Set_3). The frequent tuples in AO-Single scan algorithm are called 2-nodesets where each node represents either a concept or a value. The fourth column in the table shows the ratio

Table 8.2 No. of candidate tuples Vs. No. frequent tuples

Algorithm	No. of candidates	No. of frequent itemsets	Ratio
AO-Single scan	624	348	0.558
Data-driven algorithms	16181	1947	0.120

of frequent tuples to the number of candidates. From this column it is observed that the ratio of the frequent (useful) tuples generated to the number of candidates generated by the AO-single scan algorithm is more than that of data-driven algorithms.

Observation : *Knowledge based schemes like AO-single scan algorithm generate less number of candidates and require less number of database scans when compared with the data-driven mining algorithms.*

8.1.3 Multi-database Based Schemes

Since mining multiple databases for associations is not reported in the literature, we compare *only* the schemes reported in this thesis. The algorithms considered are :

- AMAAM (Extended inverted file based algorithm) [Chapter 4],
- AO-Single scan (A O-taxonomy based single scan algorithm)[Chapter 5],
- PC-tree based algorithm (MMUP) [Chapter 6], and
- MDLN-structure based algorithm [Chapter 7].

Table 8.3 shows a comparison based on the characteristics of the ARM activity. The

Table 8.3 Characteristics of ARM activity

Characteristics	Algorithms			
	AMAAM	MDLN Based	AO-Single scan	MMUP
Based on semantic modeling	Yes	Yes	Yes	Yes
Distributed mining capability	No	Yes	No	Yes
Dynamic/incremental mining capability	No	Yes	No	Yes
Valuetable	Implicit	Implicit	Explicit	Implicit

parameters considered here are :

1. Semantic modeling : That is, use of domain knowledge for mining associations. All the proposed algorithms use domain knowledge in the form of a semantic network to identify the relevant databases.

2. Distributed mining : That is, capability of the algorithm to handle distributed mining. MDLN-structure and the PC-tree based algorithms can be used for distributed mining.

3. Dynamic/incremental mining capability : The MDLN-structure based algorithm is capable of handling change of data (incremental mining), whereas the PC-tree based algorithm is capable of handing change of data, knowledge and user defined minimum support value (dynamic mining).

4. Valuetable : AMAAM, MMUP, and MDLN-structure based algorithms generate valuetable implicitly while building structures. AO-Single scan algorithm needs valuetable, which is required as input and so it is created explicitly.

Now, we compare these algorithms based on : number of candidate tuples generated, time and space complexity of algorithms.

Notations used :

N - number of databases

m - number of values in the relation of reference

n - maximum number of tuples in any relation

p - number of partitions

T - height of AO-taxonomy

r : fanout of any node in AO-taxonomy

8.1.3.1 Number of Candidate Tuples

A candidate tuple of a mining algorithm is a potential frequent tuple. Depending on how the candidate tuples are handled, we have two cases:

1. generate and test : Some algorithms generate potential candidate tuples without referring to the database and test them with the help of database. AMAAM algorithm, AO-Single scan algorithm are examples of this category.

2. test : Some algorithms will not generate any candidates but only test them. MMUP and MDLN-structure based algorithm are examples of this category.

Table 8.4 shows the possible order of algorithms based on the number of candidate tuples generated.

Table 8.4 A comparison table : Total number of candidate tuples

Algorithm Name	No. of Candidate tuples	
	Best case	Worst case
AMAAM	$\bigcirc(m.p)$	$\bigcirc(p.2^m)$
AO-Single scan	$\bigcirc(1)$	$\bigcirc(r^{2T})$
MMUP	$\bigcirc(m)$	$\bigcirc(3^m)$
MDLN-structure based	$\bigcirc(m)$	$\bigcirc(2^m)$

Lemma 8.1.3.1 In the best case the number of candidate tuples,

1. generated and tested by AMAAM is $\bigcirc(m.p)$.
2. generated and tested by AO-Single scan algorithm is $\bigcirc(1)$.
3. tested by MMUP is $\bigcirc(m)$.
4. tested by MDLN-structure based algorithm is $\bigcirc(m)$.

Proof :

1. The best case for AMAAM occurs when there are no frequent 1-tuple (value). So, AMAAM has to generate and test m global-potentially frequent 1-tuple in each of the partition. So, the complexity is $\bigcirc(p.m)$.
2. In AO-taxonomy, each non-leaf node is described by concept. Each concept is a description which is set of one or more sets of values at leaf nodes of the AO-taxonomy. In the best case scenario for number of candidates generated, the AO-taxonomy has two nodes. This is because, we require at least two nodes, say N1 and N2 of AO-taxonomy to form an association rule of the form 'concept1 $\Longrightarrow$ concept2' or/and 'concept2 $\Longrightarrow$ concept1'; where concept1

and concept2 are descriptions of N1 and N2 respectively. So, the number of candidates generated and tested is 2, i.e., of the order, $\bigcirc(1)$.

3,4. In MMUP and MDLN-structure based algorithm, best case corresponds to having no frequent 1-tuple (value). So, at the end of one database scan we can find out that there are no frequent 1-tuples (values). So, further processing stops. This requires considering $\bigcirc(m)$ tuples. □

Lemma 8.1.3.2 The number of candidate tuples, in the worst case

1. generated and tested by AMAAM is $\bigcirc(p.2^m)$.
2. generated and tested by AO-Single scan algorithm is $\bigcirc(2^{rT})$
3. tested by MMUP is $\bigcirc(3^m)$.
4. tested by MDLN-based structure is $\bigcirc(2^m)$.

Proof

1. The worst case for AMAAM is when every frequent k-tuples, $k = 1, 2, \ldots, m$ is present in every partition. So, the number of globally frequent tuples per partition is $2^m - 1 = \bigcirc(2^m)$. So, the number of candidate tuples is $\bigcirc(p.2^m)$.
2. Under worst case, AO-taxonomy is a balanced tree. So, the worst case for number of candidate 2-nodesets (similar to tuple) is $\bigcirc(r^{2T})$.
3. The worst case for MMUP is, the target PC-tree have a situation where all the possible paths exists. In a PC-tree, there are $\binom{m}{i}$ paths of length $i = 1, 2, \ldots, m$. So, the total number of paths is 2^m. Each path length m corresponds to at most $\bigcirc(2^m)$ candidates. So, the worst case number of candidate tuples is $\binom{m}{m} \times 2^m + \binom{m}{m-1} \times 2^{m-1} + \ldots + 1 \times 2^0 = \bigcirc(3^m)$.
4. In the worst case, the MDLN structure stores m values. So, the number of candidates tested is $\bigcirc(2^m)$. □

8.1.3.2 Space and Time Complexity

Mining algorithms store abstractions in the form of candidate tuples or structures in the main memory. For example, scheme based on EIF stores candidate tuples in main memory and AO-Single scan, MMUP, and the scheme based on MDLN-structure store the structure in the main memory. Let us first consider the space complexity[Cormen et.al 98] of the proposed algorithms. Space is required to store both the candidate tuples and the structures generated by the algorithms. The best case and the worst case space complexities of these algorithms are as shown in table 8.4. Table 8.5 shows the worst case time complexity of the proposed algorithms.

Table 8.5 A comparison table : Time complexity

Algorithm name	Worst case time complexity
AMAAM	$\bigcirc(N.n.2^m)$
MMUP	$\bigcirc(N.n) + \bigcirc(3^m)$
MDLN-structure based	$\bigcirc(N.n) + \bigcirc(2^m)$
AO-Single scan	$\bigcirc(N.n) + \bigcirc(r^{2T})$

Explanation for Table 8.5 : The worst case time complexity for the proposed algorithm is due to the number of candidate tuples or generation of the structure. The term $N.n$ in $\bigcirc(N.n.2^m)$ for AMAAM is due to N databases each with n number of tuples. The term $\bigcirc(N.n)$ for AO-Single scan / MMUP /MDLN-structure based scheme corresponds to scanning N databases each with n tuples.

Observation : *From Tables 8.3 to 8.5 it is clear that among the algorithms considered for mining multiple databases, MDLN-structure based algorithm needs a single scan of each database, supports n-ary associations and requires less space (worst case - $\bigcirc(2^m)$) and time (worst case - $\bigcirc(N.n) + \bigcirc(2^m)$) for generating associations among the proposed data-driven algorithms.*

8.1.4 Single Database Based Schemes

In this section we compare the single database based schemes. We take one or more representative algorithms from each of the categories discussed in section 2.6 based on their

popularity and versatility along with our contributions for comparison. We have chosen Apriori, Partition based and FP-tree based algorithms among the existing algorithms. We consider our schemes - PC-tree and its variants - MPC and GPC based algorithms [Chapter 6], FP-tree variants - MLOFP and GLOFP based algorithms [Chapter 6] and AO-Single scan [Chapter 5] - for comparison because basically they are single database oriented algorithms. The parameters used for the comparison are : number of database scans, number of candidate itemsets generated, time complexity and space complexity. The representative algorithms selected are :

- Apriori algorithm,
- Partition based algorithm,
- FP-tree based algorithm,
- MLOFP/GLOFP-tree based algorithm,

  AO-Single scan algorithm, and
- PC/MPC/GPC-tree based algorithm.

Notations used :

D : transaction database having tuples $t_1, t_2, \ldots, t_n$

I : a set of items $i_1, i_2, \ldots, i_m$

m : number of items

p : number of blocks in the partition

α : The probability that the partition based algorithm terminates at the end of one database scan

β : It is the probability of termination at the end of one database scan when number of large itemsets is zero

T : height of AO-taxonomy

r : fanout of any node in AO-taxonomy

8.1.4.1 Number of Database Scans

This is one of the important parameters considered while comparing the performance of mining algorithms. Since a database scan needs several disk accesses, which is a very slow process requiring time in milliseconds when compared with main memory access that requires time in nanoseconds to microsecond range, more the number of database scans, lesser will be the efficiency of the algorithm. We have mining algorithms ranging from those which need many database scans to algorithms which need only one database scan.

Table 8.6 shows the best, worst and average case scenario in terms of number of database scans among the algorithms which we considered for this study.

Table 8.6 A comparison table : Number of database scans

Algorithm Name	Best case	Worst case	Average case
Apriori	1 (viz k=0)	m	m/2
Partition based	1 (ref. lemma 8.1.4.2)	2	2 - α
FP-tree based	1	2	2 - β
MPC/MLOFP-tree based	1	2	2 - β
GLOFP-tree based	1	2	2 - β
PC-tree based	1	1	1
GPC-tree based	1	1	1
AO-Single scan	1	1	1

We characterize the important properties of some of these algorithms in the rest of this subsection corresponding to the number of database scans.

- Corresponding to average case behaviour of Apriori algorithm, we have the following lemma.

  Lemma 8.1.4.1 The average number of database scans required by Apriori is $\frac{m}{2}$.

  Proof Number of database scans is 1, when there are no candidate 2-itemsets. This happens with a probability of ξ_1. Number of database scans is 2 when there are no candidate 3-itemsets which occurs with a probability of ξ_2. Continuing like this, the number of database scans is $(m-1)$ when there are no m-candidate itemsets, which has a probability of ξ_{m-1}. If all the probabilities

$\xi_1, \xi_2, \ldots, \xi_{m-1}$ are equal to $1/(m-1)$ then the expected number of database scans is given by

$$= 1 \times \frac{1}{m-1} + 2 \times \frac{1}{m-1} + \cdots + (m-1) \times \frac{1}{m-1}$$

$$= \frac{1}{m-1}[1 + 2 + \cdots + m - 1]$$

$$\frac{m}{2}. \qquad \square$$

- The partition algorithm [Savasere et al., 95] is terminated if all the globally frequent itemsets are found. It is possible to find all globally frequent itemsets at the end of the first database scan itself. We characterize this behaviour using the following formalism :

  Let the database D be partitioned into $P_1, P_2, \ldots, P_p$, where $P_i \cap P_j = \emptyset$ $(i \neq j)$ and $\cup_{i=1}^{p} P_i = D$. Let l_i be the set of all locally-frequent itemsets in P_i. Let $s_i(x)$ be the support of itemset, x in P_i. Let $L = l_1 \cup l_2 \cup \ldots \cup l_p$.

  Definition 8.1.4.1 Total Local support (TLS) : Let $x \in L$.

  $\mathrm{TLS}(x) = \sum_{i=1}^{p} s_i(x)$.

  Lemma 8.1.4.2 The partition algorithm can be terminated at the end of one database scan iff

  $$\forall x (x \in L \Longrightarrow \mathrm{TLS}(x) \geq \sigma) \tag{8.1}$$

  Proof We know that partition algorithm will terminate if a complete set of globally-frequent itemsets, say G is found. It is given that, $L = l_1 \cup l_2 \cup \ldots \cup l_p$. If the condition (8.1) is satisfied, then $L = G$. Hence algorithm can terminate and hence the proof. $\square$

8.1.4.2 Number of Candidate Itemsets

A **candidate itemset** of a mining algorithm, $\mathbb{A}$, is a potentially large itemset that is generated and tested by $\mathbb{A}$. This process of generation and test is carried out in two ways:

1. The generation of the potential itemset is done without referring to the database and testing is done with the help of the database. Algorithms based on Apriori, Partition, AO-Single scan belong to this category.

2. Generation of the potential itemset is done using the database and testing is done without using the database explicitly. Algorithms based on FP-tree and PC-tree are examples of this category.

The number of candidate itemsets generated is database and support value dependent. Table 8.7 shows the possible order of algorithms based on the number of candidate itemsets generated.

Table 8.7 A comparison table : Total number of candidate itemsets

Algorithm Name	No. of Candidate itemsets	
	Best case	Worst case
Apriori	$\bigcirc(m)$	$\bigcirc(2^m)$
Partition based	$\bigcirc(m.p)$	$\bigcirc(p.2^m)$
FP-tree based	$\bigcirc(m)$	$\bigcirc(2^m)$
MLOFP-tree based	$\bigcirc(m^2)$	$\bigcirc(2^m)$
GLOFP-tree based	$\bigcirc(m)$	$\bigcirc(2^m)$
PC-tree based	$\bigcirc(m)$	$\bigcirc(3^m)$
MPC-tree based	$\bigcirc(m^2)$	$\bigcirc(3^m)$
GPC-tree based	$\bigcirc(m)$	$\bigcirc(3^m)$
AO-Single scan	$\bigcirc(1)$	$\bigcirc(r^{2T})$

Lemma 8.1.4.3 In the best-case, the number of candidate itemsets

1. generated and tested by Apriori is $\bigcirc(m)$.

2. generated and tested by Partition-based algorithm is $\bigcirc(m.p)$.

3. tested by FP/GLOFP-tree based algorithm is $\bigcirc(m)$.

4. tested by MLOFP/MPC-tree based algorithm is $\bigcirc(m^2)$.

5. tested by PC/GPC-tree based algorithm is $\bigcirc(m)$.

6. generated and tested by AO-Single scan algorithm is $\bigcirc(1)$.

Proof

1,2,3: The best case for all the three occurs when there are no large 1-itemsets. Apriori has to generate and test m potentially large 1-itemsets, so the complexity is $\bigcirc(m)$. Partition-based algorithm has to generate and test m locally-potentially large 1-itemsets in each partition. So, the complexity is $\bigcirc(p.m)$. In the case of FP/GLOFP-tree based algorithms, FP/GLOFP-trees are not constructed when there are no large 1-itemsets. However, all 1-itemsets have to be examined for possible frequent itemsets. So, the number of candidates is $\bigcirc(m)$.

4: In the case of MLOFP/MPC-tree based algorithms, MLOFP/MPC-trees are not constructed when there are no large 2-itemsets. However all 2-itemsets have to be examined for possible frequent itemsets. So, the number of candidates is $\bigcirc(m^2)$.

5: In PC/GPC-tree based algorithms, best case corresponds to having no large 1-itemsets. So, at the end of one database scan we can find out that there are no large 1-itemsets. So, further processing stops. This requires considering $\bigcirc(m)$ itemsets.

6: In AO-Single scan, best case corresponds to having two nodes in the AO-taxonomy (reason for two nodes is that it is the minimum number of elements required to construct an association rule). So, best case complexity is $\bigcirc(1)$.

□

Lemma 8.1.4.4 The number of candidate itemsets, in the worst case

1. generated and tested by Apriori is $\bigcirc(2^m)$.
2. generated and tested by Partition-based algorithm is $\bigcirc(p.2^m)$.
3. tested by the FP/MLOFP/GLOFP-tree based algorithm is $\bigcirc(2^m)$.
4. tested by the PC/MPC/GPC-tree based algorithm is $\bigcirc(3^m)$.
5. generated and tested by the AO-Single scan algorithm is $\bigcirc(r^{2T})$.

Proof

1. This happens when every $x \in C_k, k = 2, 3, \ldots, m$ is a large itemset, where C_k - set of candidate k-itemsets. The number of candidate itemsets generated and tested is $\bigcup_{k=2}^{m} C_k \cup l_1 = 2^m - 1$, which is $\bigcirc(2^m)$.
2. The worst case scenario is when every large k-itemset, $k = 1, 2, \ldots, m$ is present in every locally-large itemset,$ll_i, i = 1, 2, \ldots, p$. So, the number of locally-large itemsets per partition is $2^m - 1 = \bigcirc(2^m)$. So, the number of candidate itemsets is $\bigcirc(p.2^m)$.
3. The worst case for FP/MLOFP/GLOFP-tree correspond to a situation where all the possible paths exists. Note that frequency of each large 1-itemset is the same. So, there are m paths in the FP-tree. So, the number of candidate itemsets tested by the algorithm is $\bigcirc(2^m)$, because each path has at most $\bigcirc(2^m)$ candidate itemsets.
4. Refer to proof (3) of lemma 8.1.3.2.
5. Refer to proof (2) of lemma 8.1.3.2. □

8.1.4.3 Space and Time Complexity

Mining algorithms store abstractions generated using the preprocessor in the main memory as discussed in section 2.7. Also, algorithms like the one based on PC-tree or its

variants need to store the PC-tree structure in the main memory. Here, we consider first the space complexity [Cormen et al., 98] of the chosen algorithms. Space is required to store either the candidate itemsets or the tree structure generated by the algorithm. The best case and the worst case space complexities of these algorithms are as shown in Table 8.7. Only exception is the Apriori algorithm. Even though the total number of candidate itemsets generated by Apriori under worst case is $\bigcirc(2^m)$, at most $\bigcirc(\binom{m}{m/2})$ is the space required for storing candidate itemsets during any iteration.

Table 8.8 shows the worst case time complexity of the algorithms short-listed.

Table 8.8 A comparison table : Time complexity

Algorithm name	Worst case time complexity
Apriori	$\bigcirc(n.m.2^m)$
Partition based	$\bigcirc(n.2^m)$
FP-tree based	$\bigcirc(n) + \bigcirc(2^m)$
MLOFP/GLOFP-tree based	$\bigcirc(n) + \bigcirc(2^m)$
PC/MPC/GPC-tree based	$\bigcirc(n) + \bigcirc(3^m)$
AO-Single scan	$\bigcirc(n) + \bigcirc(r^{2T})$

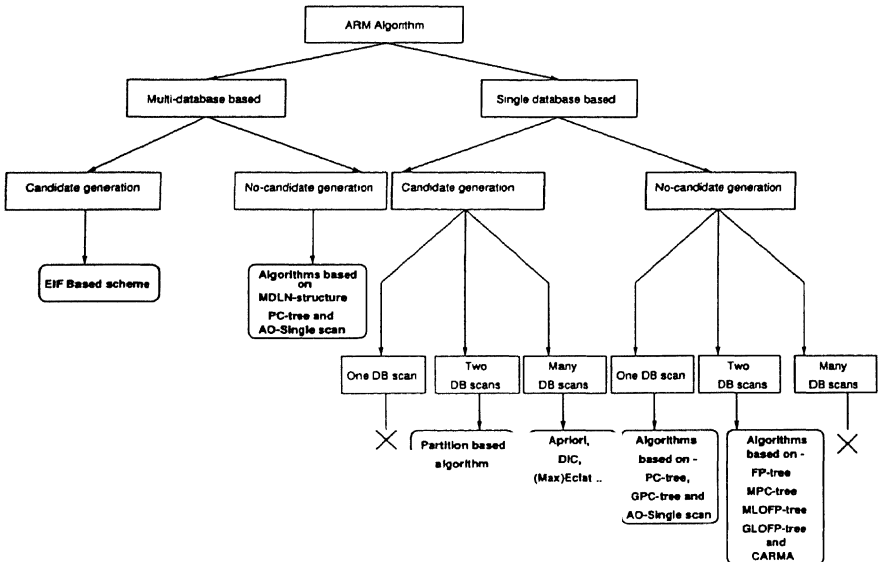
Explanation for Table 8.8 : The worst case time complexity for the algorithms listed, is due to the number of candidate itemsets generated under worst-case shown in Table 8.7. The term $n.m$ in $\bigcirc(n.m.2^m)$ for Apriori is due to n tuples in the database with each tuple having at most m values. The term n in $\bigcirc(n.2^m)$ for Partition based algorithm is due to n tuples in the database. The term $\bigcirc(n)$ for FP/MLOFP/GLOFP/PC/MPC/GPC is to scan the database of n tuples. The term $\bigcirc(n)$ for AO-Single scan is the time complexity to scan the database.

Observation : *Among all single database based algorithms, the AO-Single scan algorithm is the* best *algorithm because (i) it scans the database only* once *(ii) number of candidate itemsets generated and tested is less (worst case - $\bigcirc(r^{2T})$) (iii) space (worst case - $\bigcirc(r^{2T})$) and time (worst case - $\bigcirc(N.n) + \bigcirc(r^{2T})$) complexity of the algorithm is also* less.

Figure 8.4 depicts the taxonomy of ARM algorithm based on multiple database based

and single database based mining. Nodes marked by X in the tree indicate that there are no representative algorithms belonging to that category.

Figure 8.4 Taxonomy of ARM algorithms



8.2 Summary

In this chapter, we have compared the multi-database based and single database based ARM algorithms based on several important parameters.

Identification of relevant databases for multi-database based mining is done with the help of knowledge in the form of a semantic network. We used schema integration methodology for generating the navigation path across the identified databases. Next, we briefly discuss the salient features of the schemes proposed in this thesis.

Extended inverted file based scheme discovers the associations between set of values of attributes belonging to different databases. This scheme : (i) semantically partitions the resultant database; so, association rule generation is a single step process, and (ii) is multi-database oriented.

A O taxonomy based scheme discovers the associations between values or concepts

which are given as input in the form of knowledge tree. The AO-Single scan scheme : (i) discovers interesting association rules, (ii) performs interactive and relevant association rule mining, (iii) scans the relation of reference only once, (iv) is single database oriented, and (iv) generates small sized abstractions among all the schemes considered.

Pattern count tree based scheme generates a structure which is a compact and a complete representation of the database. This scheme : (i) discovers all association rules, (ii) is ideally suited for dynamic mining, (iii) scans the relation of reference only once, and (iv) is single database oriented.

MDLN-structure based scheme generates n-ary associations based on dominant entity. This scheme : (i) discovers all associations which are not based on co-occurrence of values in the tuple, (ii) scans each relevant database for mining only once, (iii) generates compact abstractions among all data-driven schemes considered, (iv) is multi-database oriented, and (iv) supports incremental mining.

The conclusion of our study are :

- The most important parameter affecting the performance of association rule mining algorithms is the number of database scans. Most of the algorithms proposed and discussed in the thesis scan the relation of reference (or valuetable) which is the target database only once.

- Another important parameter affecting the performance of association rule mining algorithms is the type of knowledge used. Even though, *is_a* taxonomy is the most frequently used knowledge structure for mining generalized association rules, we have demonstrated, in the thesis, that by using richer knowledge structures like AO taxonomy and semantic network, one can obtain more interesting and useful association rules.

- Among all multi-database based algorithms, MDLN-structure based algorithm is the *best* algorithm because, it needs a single scan of each of the databases for structure construction, supports n-ary associations, and requires less space (worst case - $\bigcirc(2^m)$) and time (worst case - $\bigcirc(N.n)+\bigcirc(2^m)$) for generating associations

among all data-driven algorithms.

- AO taxonomy based mining algorithms can effectively reduce the number of candidate tuples. The value of this parameter can vary from exponential order in the number of tuples, in the case of data-driven algorithms, to $\bigcirc(1)$ in the case of AO taxonomy based single scan mining algorithm.

Future work :

Some of the important directions that need to be pursued to make ARM activity more meaningful are :

- Issues in generating frequent tuples are - the time required to convert candidate tuples to frequent tuples; and the space required to store the candidate tuples. Developing parallel algorithms and running the existing algorithms on parallel processors is to be studied more seriously for increased scale-up, size-up and speed-up [Zaki et.al, 99B].

- It is natural to assume that data in the databases and knowledge are dynamic, that is they change with time. However, most of the mining algorithms are designed to handle static data and knowledge. Incremental mining algorithms can handle changing data only. However, no work is reported on mining data when data and knowledge are changing. PC-tree based algorithm is an exception. The notion of dynamic mining is very practical. So it has to be studied further.

- Interactive mining is an important pragmatic direction, where user's inputs are integrated into the process of mining. More research work is needed in this area to mine for meaningful association rules.

- Structure based abstractions seem to offer a wide variety of benefits. In the thesis, their usage in ARM algorithms is examined. It could be interesting to study their role in other data mining activities also. In this thesis, we assumed that the structures fit into the main memory. The proposed schemes need to be extended to handle situations where the structures generated do not fit into the main memory.

Applications : There are several important application areas. One of the most important areas is **web mining**. The structures discussed in the thesis use well-structured data. However, use of these structures on semi-structured data like the web documents will be useful in knowledge based information retrieval.

Bibliography

[1] Adriaans, P., Zantinge, D. 1999. *Data Mining*, Addison - Wesley.

[2] Aggarwal, C.C., Philip, S.Yu. 1997. *Online generation of association rules*, Technical Report RC20899 (92609), IBM Research Division, T.J. Watson Research Center, NY.

[3] Agrawal, R., Ghosh, S., Imielinski, T., Iyer, B., Swami, B. 1992. *An interval classifier for database mining applications*, Proc. of the 18^{th} VLDB Conf., pg : 560 - 573.

[4] Agrawal, R., Imielinski, T., Arun Swami. 1993 (A). *Database mining: A performance perspective*, IEEE Transactions on Knowledge and Date Engg. pg : 914 - 925.

[5] Agrawal, R., Imielinski, T., Arun Swami. 1993 (B). *Mining association rules between sets of items in large databases*, Proc. of ACM SIGMOD. pg : 207 - 216.

[6] Agrawal, R., Faloutsos, C., Arun Swami. 1993 (C). *Efficient similarity search in sequence databases*, Proc. of the fourth Int'l Conference on Foundations of Data Organization and Algorithms.

[7] Agrawal, R., Srikant, R. 1994. *Fast algorithms for mining association rules in large databases*, Proc. of 20^{th} Int'l Conf. on VLDB, pg : 487 - 499.

[8] Agrawal, R., Srikant, R. 1995 (A). *Mining sequential patterns*, Proc. of 11^{th} international conf. on Data Engg. pg : 3 - 14.

[9] Agrawal, R., Lin, K., Sawhney, H.S., Shim, K. 1995 (B). *Fast similarity search in the presence of noise, scaling, and translation in time-series databases*, Proc. of the 21^{st} Int'l Conf. on VLDB.

[10] Agrawal, R., Shafer, J.C. 1996. *Parallel mining of association rules : Design, implementation and experience*, IBM Research Report RJ1004.

[11] Ananthanarayana, V.S. 1995. *Schema integration in federated databases and logical database design*, M.E. Thesis, Dept. of CSA, IISc.

[12] Ananthanarayana, V.S., Narasimha Murty, M., Subramanian, D.K. 2000 (A). *Mining large itemsets using a single database scan*, IISc-CSA, Technical Report.

[13] Ananthanarayana, V.S., Subramanian, D.K., Narasimha Murty, M. 2000 (B). *Scalable, distributed and dynamic mining of association rules using PC-trees*, Int'l Conference on High Performance Computing, (Springer-Verlag Berlin Heidelberg) pg : 559 - 566.

[14] Ananthanarayana, V.S., Narasimha Murty, M., Subramanian, D.K. 2000 (C). *Mining for frequent valuesets (itemsets) : A review from the database perspective*, IISc-CSA, Technical Report IISc-CSA-2000-3.

[15] Ananthanarayana, V.S., Narasimha Murty, M., Subramanian, D.K. 2001 (A). *Multi-Dimensional Semantic Clustering of Large Databases for Association Rule Mining*, Pattern Recognition Journal, Vol. 34(4). pg : 939 - 941.

[16] Ananthanarayana, V.S., Narasimha Murty, M., Subramanian, D.K. 2001 (B). *Efficient clustering of large data sets*, Accepted for publication in Pattern Recognition Journal.

[17] Ananthanarayana, V.S., Narasimha Murty, M., Subramanian, D.K. 2001 (C). *An incremental data mining algorithm for compact realization of prototypes*, Accepted for publication in Pattern Recognition Journal.

[18] Arun K Pujari. 2001. *Data mining techniques*, Universities press (India) Limited.

[19] Bing Liu, Hsu, W., Mun, L., Lee, H. 1999. *Finding interesting patterns using user expectations*, IEEE Trans. on Knowledge and Data Engineering, Vol.11(6). pg : 817 - 832.

[20] Brachman, R., Khabaza, T., Kloesgen, W., Shapiro, G.P., Simoudis, E. 1996. *Mining business databases*, CACM. pg : 42 - 48.

[21] Brin, S., Motwani, R., Ullman, J.D., Tsur, S. 1997. *Dynamic itemset counting and implementation rules for market-basket data*, Proc. of ACM SIGMOD. pg : 255 - 264.

[22] Brusi, C., Zeleznikow, J. 1999. *Knowledge discovery and datamining in biological databases*, The Knowledge Engineering Review, Vol. 14(3). pg : 257 - 277.

[23] Cai, C.H., Ada, W. C., Cheng, C.H., Kwong, W.W. 1998. *Mining association rules with weighted items*, Proc. of IEEE Int'l Data Engineering and Application Symposium. pg : 68 - 77.

[24] Chen, M.S., Han, J., Yu, P.S. 1996. *Data mining: An overview from database perspective*, IEEE Transactions on Knowledge and Data Engg. pg : 866 - 883.

[25] Cheung, D.W., Han, J., Ng, V.T. 1996 (A). *Maintenance of discovered association rules in large databases : An incremental updating technique*, Proc. of Int'l conference on Data Engg. pg : 106 - 114.

[26] Cheung, D.W., Lee, S.D., Benjamin Kao. 1997. *A general incremental technique for maintaining discovered association rules*, Proc. of the 5^{th} Int'l conf. on Database Systems for Advanced Applications.

[27] Cheung, D.W., Vincent, T.Ng., Ada, W.Fu., Yongjian, Fu. 1996 (B). *A fast distributed algorithm for mining association rules*, Int'l conf. on Parallel and Distributed Information System.

[28] Clark, P., Boswell, P. 1991. *Rule induction with CN2: Some recent improvements*, Machine Learning: Proc. of 5^{th} European Conf. pg : 151 - 163.

[29] Cormen, T.H., Leiserson, C.E., Rivest, R.L. 1998. *Introduction to algorithms*, PHI.

[30] Dhar, V., Tuzhilin, A. 1993. *Abstract-driven pattern discovery in databases*, IEEE Trans. on Knowledge and Data Engineering, Vol.5(6). pg : 926 -938.

[31] Duda, R.O., P.E. Hart, P.E. 1973. *Pattern Classification and Scene Analysis*, Wiley, New York.

[32] Elmasri, R., Navathe, S B. 2000. *Fundamentals of Database Systems*, Addison-Wesley Publishing Company, Third Edition.

[33] Eui-Hong(Sam) Han, Karypis, G., Vipin Kumar. 1997. *Scalable parallel data mining for association rules*, Proc. of ACM SIGMOD. pg : 277 - 288.

[34] Fakuda, T., Morimoto, Y., Morishita, S., Tokuyama. T. 1996 (A). *Data mining using two-dimensional optimized association rules : Scheme, algorithms and visualization*, Proc. of ACM SIGMOD.

[35] Fakuda, T., Morimoto, Y., Morishita, S., Tokuyama, T. 1996 (B). *Mining optimized association rules for numeric attributes*, Proc. of ACM Symposium on Principles of database systems.

[36] Faloutsos, C., Ranganathan, M., Manolopoulos, Y. 1994 *Fast sub sequence matching in time-series databases*, Proc. of the ACM SIGMOD Conference on Management of Data.

[37] Fayyad, U.M., Piatetsky-Shapiro, G., Smyth, P. 1996 (A). *The KDD Process for Extracting Useful Knowledge from Volumes of Data*, Communications of the ACM, Vol.39, No.11. pg : 27 - 34.

[38] Fayyad, U.M., Piatetsky-Shapiro, G., Smyth, P., Uthurusamy, R. 1996 (B). *Advances in Knowledge Discovery and Data Mining*, AAAI Press.

[39] Findler, N V., (Ed), 1979. *Associative Networks: Representation and use of knowledge by Computers*, Academic Press.

[40] Frawley, W.J., Piatetsky-Shapiro, G., and Matheus, C.J. 1991 *Knowledge discovery databases : An overview*, in Knowledge Discovery in Databases (Piatetsky-Shapiro G. and Frawley W.J., eds.) Cambridge, MA: AAAI/MIT, pg : 1 - 27.

[41] George M. Eberhar (compiled by), 1991. *The whole library handbook*, American Library Association (Publisher).

[42] Gupta, A., Harinarayan, V., Quass, D. 1995. *Aggregate-query processing in data warehousing environment*, Proceedings of 21^{st} Int'l Conference on VLDB, pg : 358 - 369.

[43] Han, J., Cai,Y., Cercone,N. 1992. *Knowledge discovery in databases. An attribute oriented approach*, Proceedings of 18^{th} VLDB Conference.

[44] Han, J., Cai, Y., Cercone, N. 1993. *Data driven discovery of quantitative rules in relational databases*, IEEE Trans. on Knowledge and Data Engineering, Vol.5(1). pg : 29 - 40.

[45] Han, J., Fu, Y. 1994. *Dynamic generation and refinement of concept hierarchies for knowledge discovery in databases*, AAAI'94 Workshop on Knowledge Discovery in Databases(KDD 94), pg : 157 - 168.

[46] Han, J., Fu, Y. 1995. *Discovery of multiple-level association rules from large databases*, Proc. of 21st conf. on VLDB, pg : 420 - 431.

[47] Han, J., Fu, Y. 1996. *Exploration of the power of attribute-oriented induction in data mining*, In Advances in Knowledge Discovery and Data Mining, Eds : Fayyad, U.M., Piatetsky-Shapiro, G., Smyth, P., Uthurusamy, R., AAAI/MIT Press, pg : 399 - 421.

[48] Han, J., Laks V.S Lakshmanan, Raymond T. Ng. 1999. *Constraint-based multidimensional data mining*, IEEE Computer. pg : 46 - 50.

[49] Han, J., Fu, Y. 1999. *Discovery of multiple-level association rules from large databases*, IEEE Trans. of Knowledge and Data Engineering, 11(5), pg : 1 - 8.

[50] Han, J., Pei, J., Yin, Y. 2000. *Mining frequent patterns without candidate Generation*, Proc. of ACM-SIGMOD.

[51] Hidber, C. 1999. *Online Association Rule Mining*, Proc. of ACM-SIGMOD. pg : 145 - 156.

[52] Holsheimer, Kersten, M., Mannila, H., Toivonen, H. 1995. *A perspective on databases and data mining*, 1_{st} Int'l conf. on KDD, pg : 150 - 155.

[53] Horowitz, E., Sahni, S. 1983. *Fundamentals of data structures*, CBS Publishers & Distributors, Delhi.

[54] Houtsma, M., Arun Swami. 1995. *Set oriented mining for association rules in relational databases*, Proc. of Int'l conf. on Data Engg. pg : 25 - 33.

[55] Inmon, W.H., 1996. *The data warehouse and data mining*, Communications of the ACM. pg : 49 - 50.

[56] Jain, A.K., Dubes, R.C. 1995. *Pattern recognition, course lecture notes*, Michigen State University, CAPCO.

[57] Jain, A.K., Narasimha Murty, M., Flynn, P.J. 1999. *Data Clustering : A review*, ACM Computing Surveys. pg : 264 - 323.

[58] Jun-Lin Lin, Dunham, M.H. 1998. *Mining Association Rules : Anti-Skew Algorithms*, 14^{th} Int'l conf. on Data Engineering. pg : 486 - 493.

[59] Kanneth C. Cox, Eick, S.G., Wills, G.J., Brachman, R.J. 1997. *Visual Data Mining : Recognizing Telephone Calling Fraud*, Data Mining and Knowledge Discovery, Vol. 1(2). pg : 225 - 231.

[60] Kim, M., Lu, F., Raghavan, V.V. 2000. *Automatic construction of rule based trees for conceptual retrieval*, Proceedings of SPIRE2000, ACoruna, Spain.

[61] Kivinen, J., Mannila, H. 1994. *The power of sampling in knowledge discovery*, Proc. ACM-SIGMOD-SIGART Symposium on Principles of Database Theory , PODS, pg : 77 - 85.

[62] Knobbe, A.J., Blockeel, H., Siebes, A., D.M.G.vander Wallen. 1999. *Multi-relational data mining*, CWI, Information Systems, Report INS-R9908.

[63] Knuth, D.E. 1981. *The art of computer programming*, 2^{nd} Ed., Vol. 2.

[64] Kuok, C.M., Ada Fu, Wong, M.H. 1998. *Mining fuzzy association rules in databases*, Proc. of ACM SIGMOD Record, Vol. 27, No. 1, pg : 41 - 46.

[65] Langley, P., Simon, H., Bradshaw, G., Zytkow, J. 1987. *Scientific discovery : Computational explorations of the creative process*, Cambridge, Mass : The MIT Press.

[66] Larson, J.A., Navathe, S.B. 1986. *A theory of attribute equivalence in databases with application to schema integration*, IEEE Trans. on Software Engineering, Vol.15, No.4, pg : 449 - 463.

[67] Lee, S.D., Cheung, D.W. May 1997. *Maintenace of discovered association rules : when to update*, Proc. 1997 ACM-SIGMOD Workshop on Data Mining and Knowledge Discovery.

[68] Liu, C.L. 1985. *Discrete mathametics*, McGraw Hill.

[69] Luc Dehaspe, Luc De Raedt. 1997. *Mining association rules in multiple relations*, In Proc. of the 7^{th} Int'l workshop on Inductive Logic Programming, volume 1297 of Lecture Notes in Artificial Intelligence, Springer-Verlag. pg : 125 - 132.

[70] Major, J.A., Mangano, J.J. 1993. *Selecting among rules induced from a hurricane database*, Proc. AAAI 93, Workshop Knowledge Discovery in Databases. pg : 28 - 41.

[71] Mendelson, E. 1964. *Introduction to mathematical logic*, Princeton, NJ: D.Van Nostrand.

[72] Morimoto, Y., Fakuda, T., Matsuzawa, H., Tokuyama, T., Yoda, K. 1998. *Algorithms for mining association rules for binary segmentations of huge categorical databases*, Proc. 24^{th} Very Large Databases Conf. pg : 380 - 391.

[73] Morishita, S. 1998. *On classification and regression*, Proc. 1^{th} Intl. Conf. on Discovery Science - Lecture Notes in AI. pg : 40 - 57.

[74] Mueller, A. 1995. *Fast sequential and parallel algorithms for association rule mining: A comparison*, Technical Report CS-TR-3515, Dept. of Computer Science, Univ. of Maryland, College Park, MD.

[75] Ozden, B., Ramaswamy, S., Silberschartz, A. 1998. *Cyclic association rules*, Proc. of Int'l conference on Data Engg. pg : 412 - 421.

[76] Park, J.S., Chen, M.S., Philip. S. Yu. 1995. *An effective hash based algorithm for mining association rules*, Proc. of ACM SIGMOD. pg :175 - 186.

[77] Parthasarathy, S., Zaki, M.J., Ogihara, M., Dwarakadas, S. Nov, 1999. *Incremental and Interactive Sequence Mining*, 8^{th} Int'l conf. on Information and Knowledge Management.

[78] Pei, J., Han, J., Mao, R. May, 2000. *CLOSET: An efficient algorithms for mining frequent closed itemsets*, Proc. 2000 ACM-SIGMOD Int. Workshop on Data Mining and Knowledge Discovery.

[79] Prakash, M., Narasimha Murty, M. 1997. *Growing subspace pattern recognition methods and their neural network models*, IEEE Trans. on NN, 8(1), pg : 161 - 168.

[80] Quest site : http://www.almaden.ibm.com/cs/quest

[81] Raman, S., Appan, T.S. 1998. *A variation of partition algorithm for mining association rule in very large databases*, National seminar on BPRIT, Burla, Orissa.

[82] Raymond, T. Ng, Laks V.S. Lakshmanan, J.Han, 1998. *Exploratory mining and pruning optimizations of constrained association rules*, ACMSIGMOD. pg : 13 - 24.

[83] Roberto, J.B.Jr., Agrawal, R., Gunopulos, D. 1999 (A). *Constraint-Based Rule Mining in Large, Dense Databases*, 15^{th} Int'l Conf. on Data Engineering. pg : 188 - 197.

[84] Roberto, J.B.Jr., Agrawal, R. 1999(B). *Mining the most interesting rules*, 5^{th} ACM SIGKDD Int'l Conf. on Knowledge Discovery and Data Mining. pg : 145 - 154.

[85] Sarawagi, S., Thomas, S., Agrawal, R. 1998. *Integrating association rule mining with relational database systems : Alternatives and implications*, Proc. of the ACM SIGMOD Int'l Conf. on Management of Data.

[86] Sarda, N.L., Srinivas, N.V., 1998. *An adaptive algorithm for incremental mining of association rules*, Proc. of DEXA Workshop'98. pg : 240 -245.

[87] Savasere, S., Omiecinski, E., Navathe, S. 1995. *An efficient algorithm for mining association rules in large databases*, Proc. of Int'l conf. on VLDB.

[88] Savasere, A., Omiecinski, E., Navathe, S. 1998. *Mining for strong negative associations in a large databases of customer transactions*, Proc. of Int'l conference on Data Engg. pg : 494 - 502.

[89] Schulze-Kremer, S. 1999. *Discovery in the human genome project*, Communications of the ACM, Vol.42(11). pg : 62 - 64.

[90] Shenoy, P., Haritsa, J.R., Sudarshan, S., Bhalotia, G., Bawa, M., Shah, D. 2000. *Turbo-charging vertical mining of large databases*, SIGMOD, pg : 22 - 33.

[91] Shintani, T., Kitsuregawa, M. 1998. *Parallel mining algorithms for generalized association rule with classical hierarchy*, Proc. of ACM SIGMOD. pg : 25 - 36.

[92] Silberschartz, A., Tuzhilin, A. 1995. *On subjective measures of interestingness in knowledge discovery*, I^{st} Int'l. Conf. on Knowledge Discovery and Data Mining. pg : 275 - 281.

[93] Simoudis, E. 1996. *Reality check for data mining*, IEEE Expert: Intelligent Systems and Their Applications, 11(5), pg : 26 - 33.

[94] Spath, H. 1980. *Cluster analysis : Algorithms for data reduction and classification of objects*, Ellis Horwood Limited, U.K.

[95] Srikant, R., Agrawal, R. 1995. *Mining generalized association rules*, Proc. of 21st conf. on VLDB.

[96] Srikant, R., Agrawal, R. 1996. *Quantitative association rules in large relational tables*, Proc. of ACM SIGMOD. pg : 1 - 12.

[97] Srikant, R., Vu, Q., Agrawal, R. 1997. *Mining association rules with item constraints*, Proc. of the 3^{rd} Int'l conf. on Knowledge Discovery in Databases and Data Mining.

[98] Subramanian, D.K., Ananthanarayana, V.S., Narasimha Murty, M. 1999. *Generation of inter-database association rules based on semantic networks*, Technical Report, TR No. IISc-CSA-1999-6, Dept of CSA, IISc.

[99] Subramanian, D.K., Ananthanarayana, V.S., Narasimha Murty, M. 2000. *And-Or concept taxonomies for mining generalized association rules*, International Conf. on Knowledge Based Computer Systems. pg : 70 - 81.

[100] Subramanian, D.K., Ananthanarayana, V.S., Narasimha Murty, M. 2000. *Mining Multiple Databases for Inter-Database Association Rules*, International Conf. on Knowledge Based Computer Systems. pg : 58 - 69.

[101] Thomas, S., Sreenath, B., Khaled, A., Sanjay, R. 1997. *An efficient algorithm for the incremental updation of association rules in large databases*, AAAI.

[102] Thomas, S., Chakravarthy, S. 2000. *Incremental mining of constrained associations*, International Conf. on HiPC, (Springer-Verlag Berlin Heidelberg), pg : 547 - 558.

[103] Toivonen, H. 1996. *Sampling large databases for association rules*, Int'l conf. on VLDB. pg : 134 - 145.

[104] Trembley, J.P., Manohar, R. 1980. *Discrete mathematical structures with applications to computer science*, McGraw Hill.

[105] Trembley, J.P., Sorenson, P.G. 1976. *An Introduction to Data Structures with Applications*, McGraw-Hill.

[106] Tsur, D., Ullman, J.D., Abiteboul, S., Clifton, C., Motwani, R., Nestrov, S., Rosenthal, A. 1998. *Query Flocks : Generalization of association rule mining*, Proc. of ACM SIGMOD. pg : 1 - 12.

[107] Tung, A.K.H., Lu, H., Han, J., Feng, L. 1999. *Breaking the barrier of transactions: Mining inter-transaction association rules*, Proc. 1999 Int'l Conf. on Knowledge Discovery and Data Mining (KDD 99).

[108] Pudi, V., Haritsa J., 2000. *Quantifying the utility of the past in mining large databases*, Intl. Journal of Information Systems, Vol.25(5). pg : 323 - 344.

[109] Webb, G. 1995. *OPUS: An efficient admissible algorithm for unordered search*, Journal of AI Research. pg : 431 - 465.

[110] Widom, J. 1995. *Research problems in data warehousing*, In Proc. of 4^{th} Int'l Conference on Information and Knowledge Management, pg : 25 - 30.

[111] Yan, W.P., Larson, P. *Eager aggregation and lazy aggregation*, In Proc. of 21^{st} Int'l Conference on VLDB, pg : 345 - 357.

[112] Zaki, M.J., Parthasarathy, S., Li, W., Ogihara, M. 1997 (A). *Evaluation of sampling for data mining of association rules*, 7^{th} Workshop on RIDE.

[113] Zaki, M.J., Parthasarathy, S., Ogihara, M., Li, W. 1997 (B). *New algorithms for fast discovery of association rules*, 3^{rd} Intl. Conf. on Knowledge Discovery and Data Mining.

[114] Zaki, M.J., Mitsunori Ogihara. 1998. *Theoretical foundations of association rules*, 3^{rd} SIGMOD'98 Workshop in Research Issues in Data Mining and Knowledge Discovery. pg : 71 - 78.

[115] Zaki, M.J., Hsiao, C. 1999 (A). *CHARM : An efficient algorithm for closed association rule mining*, Technical Report 99-10, Computer Science, Rensselaer Polytechnic Institute.

[116] Zaki, M.J. Dec, 1999 (B). *Parallel and distributed association mining : A survey*, IEEE Concurrency, special issue on Parallel Mechanisms for Data Mining, Vol.7, No.4, pg : 14 - 25.

[117] Zhuge, Y., Garcia-Molina, J., Hammer, J., Widom, J. 1995. *View maintenance in a warehousing environment*, In Proc. of ACM SIGMOD Int'l Conference on Management of Data, pg : 316 - 327.